RL-TR-95-238
Final Technical Report
December 1995

SEMANTIC INTEROPERATION VIA INTELLIGENT MEDIATION

SRI International

Sponsored by
Advanced Research Projects Agency

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be releasable to the general public, including foreign nations.

RL-TR-95- 238 has been reviewed and is approved for publication.

APPROVED: Joseph V Giordano

JOSEPH V. GIORDANO
Project Engineer

FOR THE COMMANDER: John A Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify Rome I aboratory/ (C3AB), Rome NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

SEMANTIC INTEROPERATION VIA INTELLIGENT MEDIATION

Xiaolei Qian
Li Gong
Robert A. Riemenschneider

Contractor: SRI International
Contract Number: F30602-92-C-0140
Effective Date of Contract: 20 August 1992
Contract Expiration Date: 20 February 1995
Short Title of Work: Semantic Interoperation via Intelligent Mediation
Period of Work Covered: Aug 92 - Feb 95

Principal Investigator: Xiaolei Qian
Phone: (415) 859-6106

RL Project Engineer: Joseph V. Giordano
Phone: (315) 330-3681

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Joseph V. Giordano, RL/C3AB, 525 Brooks Rd, Rome NY 13441-4505.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1995	Final Aug 92 - Feb 95

4. TITLE AND SUBTITLE

SEMANTIC INTEROPERATION VIA INTELLIGENT MEDIATION

5. FUNDING NUMBERS

C - F30602-92-C-0140
PE - 33140F, 61101E
PR - H767
TA - 00
WU - 03

6. AUTHOR(S)

Xiaolei Qian, Li Gong, and Robert A. Riemenschneider

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SRI International
333 Ravenswood Ave
Menlo Park CA 94025-3493

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington VA 22203-1714

Rome Laboratory/C3AB
525 Brooks Rd
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-95-238

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Joseph V. Giordano/C3AB/(315) 330-3681

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This effort presents a query mediation approach to the trusted interoperation of autonomous databases containing data that mismatch in semantics, representations, and security policies. The main contributions are a unified policy framework of mandatory access control policies, and the automated mediation of queries between databases that resolves semantic, representational,and security policy heterogeneity. Query mediation in heterogeneous legacy databases makes both the data and the applications accessing the data interoperable. Automated query mediation relieves users from the difficult task of resolving mismatches. Decoupling semantic, representational, and security policy heterogeneity improves the efficiency of automated query mediation. Trusted query mediation makes data in isolated military and civilian databases sharable, and increases data owners' confidence and willingness in the sharing. The approach provides a seamless migration path for legacy databases, enabling organizations to leverage off investments in legacy data and legacy applications.

14. SUBJECT TERMS

Database interoperability, Database security, Distributed database management systems

15. NUMBER OF PAGES

136

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Contents

I Query Mediation 1

1 Introduction 3
- 1.1 Problem . . . 3
  - 1.1.1 Semantic Mismatch . . . 3
  - 1.1.2 Representational Mismatch . . . 4
  - 1.1.3 Security Mismatch . . . 4
  - 1.1.4 Autonomy . . . 5
- 1.2 Overview of this Part . . . 5

2 The Database Interoperation Problem: An Example 6

3 Approaches to Interoperation 9

4 Mediation Architecture 12
- 4.1 Components . . . 12
- 4.2 How Queries Are Mediated . . . 13
- 4.3 Discussion . . . 15

5 Query Mediation 18
- 5.1 Schemas and Databases . . . 18
- 5.2 Properties of Query Mediation . . . 20
- 5.3 Meaning of Query Mediation . . . 20
- 5.4 Semantics of Query Mediation . . . 21

6 A Prototype Mediator 23
- 6.1 Wrappers . . . 24
- 6.2 Translators . . . 24
  - 6.2.1 Step 1: Parsing . . . 24
  - 6.2.2 Step 2: Representation Minimization . . . 24
  - 6.2.3 Step 3: Translation to Logic . . . 25
- 6.3 Query Transformer . . . 26
  - 6.3.1 Step 1: Logical Simplification . . . 26
  - 6.3.2 Step 2: Representation Transformation . . . 26

6.4 Multiple Databases 28
6.5 Prototype Implementation 29

7 Related Work **30**

8 Conclusion **32**

II A MAC Policy Framework **33**

9 Introduction **35**
9.1 Problem 35
9.2 Overview of This Part 36

10 A Policy Framework **38**
10.1 A Model-Theoretic Formulation of Multilevel Relational Databases 38
10.2 MAC Policy 39

11 Relational Model **42**
11.1 Basic Notations 42
11.2 Atomic Decomposition 44
11.3 Information Content 45

12 Interpretation Policy **48**
12.1 Tuple-Level Labeling 48
12.2 Element-Level Labeling 50
12.3 Design Trade-Off 52

13 View Policy **54**
13.1 Sample View Policies 54
13.2 Validity and Views 55
13.3 Validity Checking 57
13.4 Validity Enforcement 58

14 Update Policy **60**
14.1 Sample Update Policies 60
14.2 Polarity and Force 61
14.3 Labeling Constraints 62
14.4 Static Inference Channels 63
14.5 Dynamic Inference Channels 64
14.6 Eliminate Inference Channels 65
14.7 Design Guidelines 68

15 Secure Interoperation 69
15.1 Principles of Secure Interoperation 70
15.2 System Model and Terminology 70
15.3 Complexity 73
15.4 Composability 75

16 Conclusion 77

III Appendix: Prototype System Design and Implementation 79

17 Introduction 81

18 Prototype Functionality 82
18.1 Query Translation 82
18.2 Query Transformation 83
18.3 Table Translation 85

19 Prototype Software 86

20 Examples 87
20.1 First Example: Basic Query Transformation 87
20.2 Second Example: Reversed Roles 88
20.3 Third Example: Split Query, Join Tables 89
20.4 Fourth Example: Auxiliary Queries 89

21 Lessons Learned in Prototype Development 93

22 Transcript of Demonstration 95

Part I

Query Mediation

Chapter 1

Introduction

SRI International has completed a research program that produced a new and unique approach to the trusted interoperation of autonomous heterogeneous databases containing data that mismatch in semantics, representations, and security policies.

1.1 Problem

The interoperation of heterogeneous databases is a pressing need today as organizations attempt to share data stored in legacy databases. These databases are independently developed and maintained to each serve the needs of a single organization. The exchange of data between such databases could be problematic not only because of differences in the representation (syntax) of data but also because of often subtle differences in the intended interpretation (semantics) of data. Thus, although translators could be constructed to reformat data from one representation to another, such a translation does not guarantee that the combined, translated data are meaningful — we could be attempting to compare apples with oranges. When we try to interoperate multilevel secure databases having different security semantics (e.g., element-level vs. tuple-level labeling, treatment of polyinstantiation) with system-high legacy databases, these same problems arise in dealing with the syntax and semantics of security.

Currently there is no technology that adequately addresses these problems. Any potential solution must address the critical issues described here.

1.1.1 Semantic Mismatch

Heterogeneity in the semantics of data arises naturally. The semantic differences are caused by the diverse needs of applications. Moreover, the relationships between heterogeneous data could be incomplete or uncertain. Examples of *semantic mismatches* are

- **Scope.** Tests for the database in a hospital laboratory includes only single tests (e.g., sodium), whereas for the database in an insurance company they also include panels (e.g., electrolyte panel), which are collections of tests.

- **Temporal Basis.** The database in a clinic records the transaction time for outpatients' visits, while the database in a hospital records the admission and release time for inpatients.

Direct comparison and combination of data with such semantic mismatches would be meaningless. The interoperation should be semantically meaningful, in terms of both the semantics of individual databases and the semantic relationships between them.

1.1.2 Representational Mismatch

In addition to semantic mismatches, the same data could be represented in various incompatible structures, and the same structure could be used to represent data with incompatible semantics [50]. The representational differences are caused by the need to bind data to representations that are most natural and efficient with respect to specific applications. In general, there simply does not exist a universal representation that is perfect for every application [5, 15, 34]. Examples of *representational mismatches* are

- **Identification.** Patients could be identified by patient id numbers in the hospital, but by social security numbers (SSNs) in the insurance company. The nature of operations in these two organizations demands that different identifiers be used, since patient data are most likely accessed by SSNs, not patient id numbers, in the insurance company.

- **Biased View.** The relationship between patients, the drugs they are allergic to, and the description of the symptoms could be represented as one relation, or in two relations where the second relation captures physicians' notes. It is impractical to represent the relationship in all possible structures.

1.1.3 Security Mismatch

The interoperation of heterogeneous military and civilian databases could be further complicated by the fact that data must be protected from untrusted access. The security policies of these databases could mismatch in many ways, examples of which are

- **Label.** The same data is classified as TOP-SECRET in one database, but as TOP-SECRET OUTER-SPACE in another.

- **Security Representation.** One database employs element-level classification, while another chooses tuple-level classification.

- **Security Semantics.** Classifying attribute CARGO means in one database that unauthorized users should not know about which flights carry what cargos, but means in another that unauthorized users should not know about what cargos are shipped to which destinations.

Without trusted semantic interoperation, either data in isolated military and civilian databases will remain inaccessible, or users will run into the risk of unauthorized access to their data through inference channels.

1.1.4 Autonomy

Because of the diverse needs of autonomous organizations, heterogeneity will persist rather than disappear. To support the interoperation of autonomous heterogeneous databases containing data with semantic, representational, and security policy mismatches, five critical issues must be addressed:

- **Semantics**. Interoperation should be semantically meaningful, in terms of both the semantics of individual databases and their semantic relationships.
- **Autonomy**. Database autonomy should be respected and preserved. Users should not be required to switch to new query languages or new schemas to access data in multiple databases.
- **Automation**. Interoperation should be automated. Users should not be required to manually resolve all the semantic and representational mismatches to access data in multiple databases.
- **Efficiency**. Automated interoperation should be computationally efficient. In particular, it should not require expensive mechanisms such as theorem-proving in higher-order logics.
- **Security**. Any access permitted within an individual database must also be permitted under trusted interoperation. Any access not permitted within an individual database must be also denied under trusted interoperation.

1.2 Overview of this Part

We present a query mediation approach to the interoperation of autonomous heterogeneous databases containing data with semantic and representational mismatches [38]. We develop a mediation architecture of interoperation that facilitates query mediation, and formalize the semantics of query mediation. Queries are mediated between multiple databases, and users or applications of a local database access data in multiple databases using the local language and schema, making both the data and the applications accessing the data in legacy databases interoperable. Queries are automatically mediated, relieving users from the difficult task of resolving semantic and representational mismatches. Semantic heterogeneity is separated from representational heterogeneity by representation minimization techniques, reducing the space of heterogeneity, and improving the efficiency of automated query mediation.

Chapter 2

The Database Interoperation Problem: An Example

Imagine a typical database user, U. User U knows a great deal about the information stored in her database, including the details of how it is stored. If the database is relational, then U is familiar with the schemes of the relations, understands the semantics of the attributes that appear in those schemes, and knows the details of the way values of those attributes are represented. U also knows how to write tolerably efficient queries in the appropriate query language—say, SQL—for retrieving data of interest to her. But U's knowledge of databases is limited to the contents of her local database and how to deal with it. There may be other databases containing data that are quite relevant to her interests, but she is most likely unaware of their existence, and almost certainly lacks the skills required to retrieve and interpret that data.

The goal of our research is to help that typical user U retrieve nonlocal data, without requiring her to acquire any additional knowledge or skills. In fact, we aim to change her way of working as little as possible. An example, which we will build upon throughout this report, should help make this point clearer.

Suppose that our database user U is a physician working at a medical clinic. Recently, she has observed several unusual allergic reactions to an experimental drug, XD2001, being tested by the clinic. The reactions are unusual in that the patients who experienced them had been using the drug for a considerable length of time without a reaction prior to experiencing the recent symptoms. Her local database A_C contains information about patients' allergic reactions, so she decides to search for other cases of unusual reactions to XD2001 that were caused by treatments this year. Three relations in the database are relevant. The relation PATIENTS_ALLERGY records incidents of allergic reactions to drugs, the relation NOTES holds any notes entered by the attending physician, including whether an allergic reaction was unusual or unexpected, and the relation PATIENTS records the time at which the patient was treated, from which the time of the allergic reaction can be roughly inferred.[1] The schemes of those relations are

[1]These relation schemes were derived from the THelper-II database, which was developed by Stanford Medical School. Note in particular that not including TRANSACTION_TIME as part of the key for PATIENTS is not a typographical error. Evidently, the intention is to use a unique id for each visit, so as to enhance confidentiality by

PATIENTS	PATIENT_ID	TRANSACTION_TIME

PATIENT_ALLERGY	PATIENT_ID	DRUG_NAME	NOTE_ID	START_TIME	···

NOTES	NOTE_ID	TEXT

where the meaning of the attributes should be largely self-explanatory. (Boldface type indicates a primary key for each relation.) An SQL query Q_C on database A_C that will retrieve the desired information is

```
SELECT PATIENT_ALLERGY.PATIENT_ID, TEXT
FROM   PATIENT_ALLERGY, PATIENTS, NOTES
WHERE  PATIENT_ALLERGY.PATIENT_ID = PATIENTS.PATIENT_ID
AND    PATIENT_ALLERGY.NOTE_ID = NOTES.NOTE_ID
AND    DRUG_NAME = 'XD2001'
AND    TIMESTAMP '1994-01-01 08:00:00' < TRANSACTION_TIME;
```

Unknown to our physician U, a research hospital that is also testing the drug has observed similar unusual reactions and recorded the fact in its database A_H. Of course, A_H is organized somewhat differently than A_C, uses somewhat different value representations, and contains somewhat different information about the events, so query Q_C could not be used to retrieve the relevant data from A_H. Rather than storing physicians' notes about allergies in a separate relation, the DRUG_ALLERGY relation in A_H contains a text field. Also, even less precise time information is available in A_H. Like A_C, A_H was designed on the assumption that the allergy data needed no time associated with it: the standard use of DRUG_ALLERGY is to check whether a patient is known to be allergic to penicillin, or sulfa drugs, or whatever, when designing a course of treatment, not to record information discovered about allergies during treatment. So, just as in A_C, the time of the observed reaction must be inferred. Rather than the relatively precise TRANSACTION_TIME stored in A_C, the only time information in A_H is the admission and release times stored in the admission records of the ADMISSIONS relation. Thus, the relevant schemes in A_H are

ADMISSIONS	PATIENT_ID	ADMISSION_TIME	PATIENT_NAME	···

DRUG_ALLERGY	PATIENT_ID	DRUG_ID	TEXT

Our physician U would be just as much interested in seeing cases where patients admitted to the hospital this year—for anyone admitted this year was certainly treated this year—exhibited an allergic reaction to XD2001, which is referred to as EXPERIMENTAL_DRUG_2001 in A_H. The query Q_H that U would like to issue on A_H, if only she had the required knowledge, is

```
SELECT DRUG_ALLERGY.PATIENT_ID, TEXT
FROM   DRUG_ALLERGY, ADMISSIONS
WHERE  DRUG_ALLERGY.PATIENT_ID = ADMISSIONS.PATIENT_ID
AND    DRUG_ID = 'EXPERIMENTAL_DRUG_2001'
AND    TIMESTAMP '1994-01-01 08:00:00' < ADMISSION_TIME;
```

making it more difficult to discover the identity of a patient via analysis of data associated with a patient id.

Although there are many problems associated with database interoperability, by *the* database interoperability problem, we mean the problem faced by U. A solution is to provide U with the relevant data from A_H when she issues the query Q_C on A_C, without requiring that she learn anything about the semantics or representation of data in A_H, or even requiring that she know that A_H exists and contains relevant data. Our approach to this solution is to provide automated support for resolving the semantic and representational mismatches—in our example, the mismatch in relation schemes, the mismatch in value representations, and the mismatch in time semantics—between A_C and A_H.

Chapter 3

Approaches to Interoperation

A database system bridges the gap between a data source and an application that accesses the data. The typical organization of a database system is shown in Figure 3.1(a), where the data are stored in a database, the database is managed by a database management system (DBMS), and the DBMS supports a query language and schema through which the application formulates queries to access the data.

The ultimate goal of the interoperation of autonomous heterogeneous databases is for multiple applications to *share* multiple data sources. There are two aspects of such sharing:

1. multiple applications sharing the same data source, and
2. multiple data sources accessed by the same application.

The centralized database approach achieves data sharing by requiring that data be stored in the same database, and that applications speak the same language and schema, as is shown in Figure 3.1(b). In terms of our scenario, clinic data (e.g., transaction time) and hospital data (e.g., admission time) must reside in the same database, and the physician at the clinic must understand the relationship between the two notions of time in order to access both data sources.

In the distributed database approach, applications still must speak the same language and schema, but data can be stored in different databases, as long as the databases are managed by the same DBMS, as is shown in Figure 3.1(c). In terms of our scenario, clinic data and hospital data can reside in two databases, but the two databases must be managed by the same DBMS, and the physician at the clinic still must understand the relationship between transaction time and admission time in order to access both data sources.

In the federated database approach, applications still must speak the same language (federated language) and schema (federated schema), but databases can be managed by different DBMSs, as is shown in Figure 3.1(d). In terms of our scenario, clinic data and hospital data can reside in two databases and be managed by two DBMSs, but the physician at the clinic still must understand the relationship between transaction time and admission time in order to access both data sources.

Naturally, the next step in the evolution is a mediated database approach, where applications can speak different languages and schemas, and databases can be managed by different DBMSs, as is shown in Figure 3.1(e). A mediator has knowledge about different languages and schemas, and

the relationships between them. When an application issues a query in one language and schema, the mediator transforms the query into other languages and schemas using its knowledge, thus enabling the application to access multiple databases without having to speak multiple languages and schemas. In terms of our scenario, the clinic and the hospital can have two separate database systems, and the mediator knows that transaction time is somewhere between admission and release times. The physician at the clinic formulates queries in the language and schema of the clinic database, which are transformed by the mediator to access both data sources. She does not have to know that the hospital database uses a different notion of time.

The biggest advantage of the mediator approach over other approaches is in supporting the interoperation of legacy databases. Users of a legacy database can access multiple databases without having to learn new languages, new schemas, or their relationships and differences. Applications built on top of a legacy database can access multiple databases without having to be recoded with queries in new languages and schemas. Thus, the mediator approach provides a seamless migration path for legacy databases, enabling organizations to leverage off investments in legacy data and legacy applications.

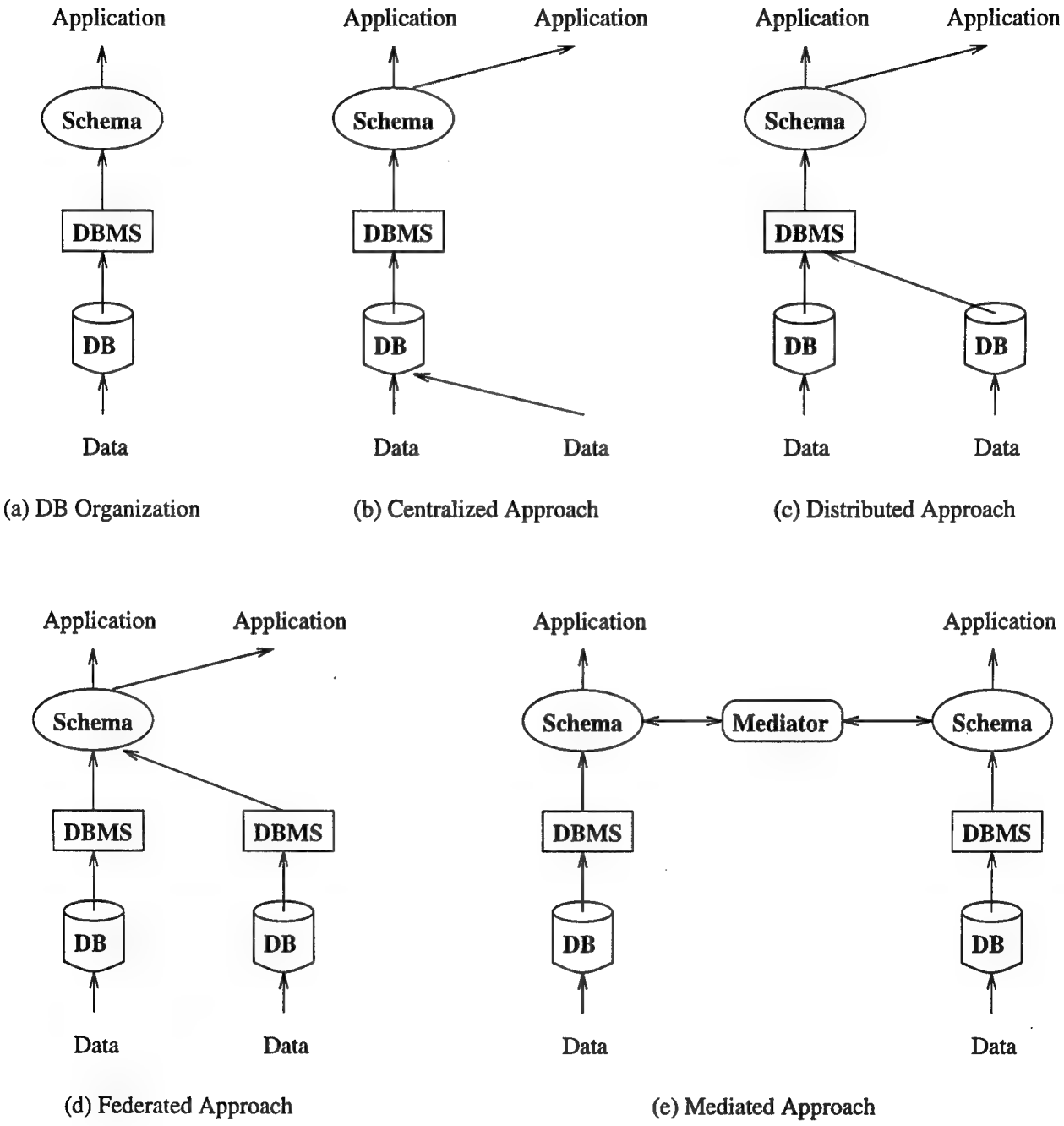


Figure 3.1: Approaches to Interoperation

Chapter 4

Mediation Architecture

The history of database evolution as depicted in Figure 3.1 demonstrates that data sharing does not necessarily mandate the sharing of system components [34] (e.g., databases, DBMSs, schemas). As long as autonomous heterogeneous databases could *communicate* with one another, they could benefit from each other's data without having to be bound to a common system component. Hence, a mediation architecture should be a communication architecture.

With the advance in semantic data models and knowledge base systems, data processing has evolved into intelligent information processing, where the availability of semantics and knowledge about data greatly enhances the capability of information abstraction. A similar evolution from data communication into intelligent information communication is essential for the interoperation of autonomous heterogeneous databases. Data should not be communicated as raw bits (i.e., syntactic communication). Instead, they should be *mediated* (i.e., semantic communication) to ensure that data from the sender will be correctly understood for processing by the receiver. Hence, a mediation architecture should be a mediator-assisted communication architecture.

4.1 Components

A mediator consists of the following components that together support the interoperation of autonomous heterogeneous databases:

- **Mediation Language.** Communication between autonomous heterogeneous databases must be carried out in a *mediation* language or *interlingua*. As pointed out in [34], this type of language differs from the representation languages (i.e., data models) of participating databases. A representation language captures the knowledge about data for the appropriate abstraction and efficient representation of one class of applications, while a mediation language captures the knowledge about data for the meaningful and efficient communication between many classes of applications.

- **Knowledge Base.** Meaningful communication between autonomous heterogeneous databases is based on the relationships between participating databases. These relation-

ships capture the commonalities and mismatches in semantics or representations between these databases. They are expressed in the mediation language and form a *knowledge base*.

- **Query Tranformer**. A mediation language alone is not sufficient to ensure meaningful communication, because autonomous heterogeneous databases might contain data that mismatch in semantics or representations. We need a *query transformer* to mediate the communication by resolving potential mismatches. Equipped with the knowledge base of relationships between participating databases, the query transformer accepts queries from one database, determines which other databases contain relevant data, generates queries to these databases, and mediates resulting data back to the original database. The mediation is carried out in the mediation language.

- **Wrappers**. Participating databases are wrapped up by *interface modules* to forward incoming queries to the query transformer, to respond to queries from the query transformer, and to receive answers from the query transformer. Wrappers are largely invisible to users, and require very little change to the interfaces of participating databases.

- **Translators**. Since the representation languages of participating databases very likely differ from the mediation language, we need *translators* to translate queries and data between these representation languages and the mediation language, in order for queries and data to be communicable by the query transformer.

- **Conflict Detectors**. When related data from multiple participating databases are merged to give answers to a query, conflicts are always possible because the merged data might be inconsistent with respect to the constraints of the original database in which the query is specified. We need *conflict detectors* to detect such potential problems. The conflict detectors support the communication of globally inconsistent but locally reasonable data [34].

Figure 4.1 shows the mediation architecture (arrows represent data flow), with three autonomous heterogeneous databases interoperating through a mediator — a *data bus*.

4.2 How Queries Are Mediated

Suppose that users issue query Q_A in database A in Figure 4.1, expressed in the language/schema of database A. The mediation of query Q_A proceeds as follows:

1. Wrapper A intercepts query Q_A, and sends it both to Query Processor A to get answer L_A and to Translator A for mediation.

2. Translator A translates query Q_A to query Q'_A expressed in the mediation language, and sends it to the Query Transformer.

3. From query Q'_A, the Query Transformer computes a mediated query Q'_B expressed in the mediation language based on its knowledge about the relationships between databases A and B, and sends it to Translator B.

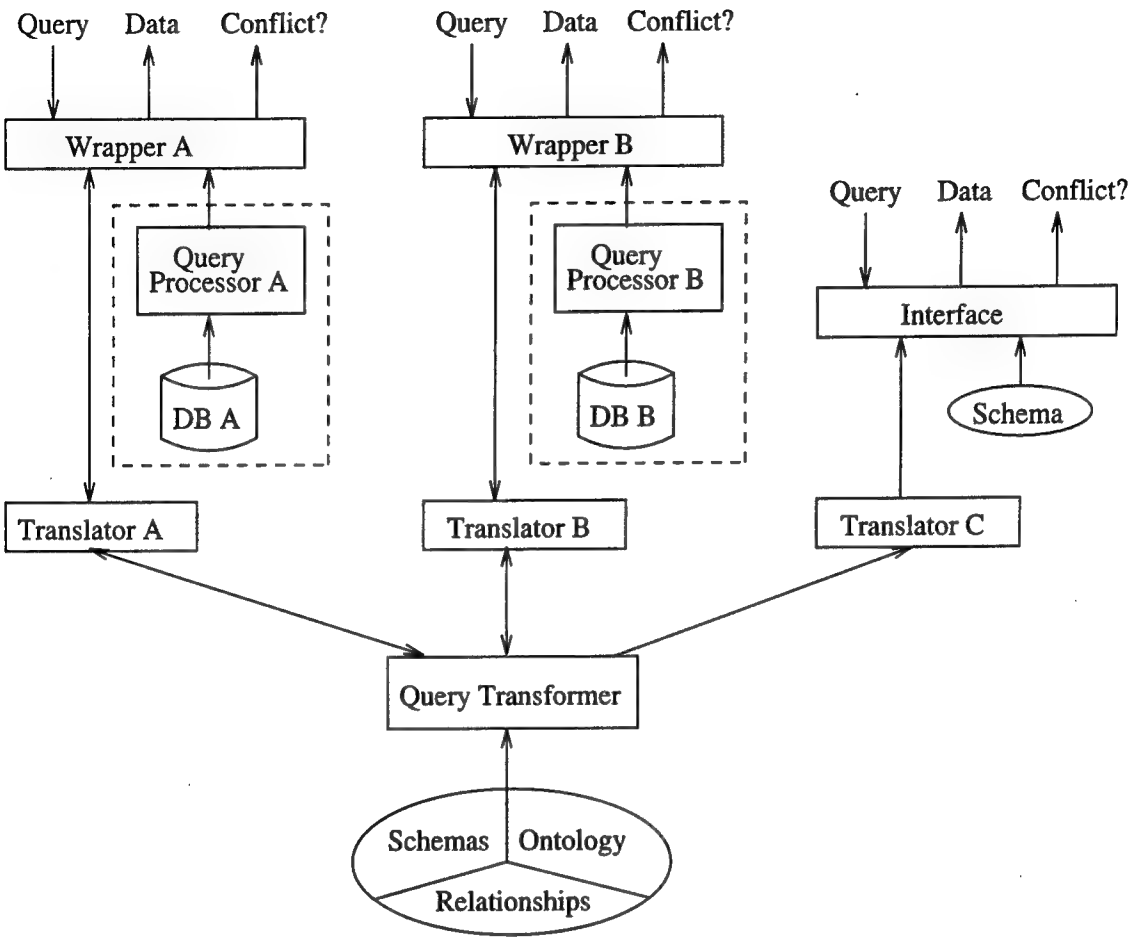


Figure 4.1: Mediation Architecture

4. Translator B translates query Q'_B to query Q_B expressed in the language/schema of database B, and sends it to Wrapper B.

5. Wrapper B sends query Q_B to Query Processor B to get answer D_B expressed in the language/schema of database B, and sends it to Translator B.

6. Translator B translates answer D_B to answer D'_B expressed in the mediation language, and sends it to the Query Transformer.

7. The Query Transformer derives answer D'_A expressed in the mediation language from answer D'_B based on its knowledge about the relationships between databases A and B, and sends it to Translator A.

8. Translator A translates answer D'_A to answer D_A expressed in the language/schema of database A, and sends it to Wrapper A.

9. Wrapper A merges answers L_A and D_A, detects conflicts in the merged answer, and presents it to users as the answer to query Q_A.

4.3 Discussion

The most important objective in designing the mediation architecture is to support the interoperation of legacy databases and legacy applications. Does the architecture of Figure 4.1 achieve this objective? The answer is yes. A legacy database (e.g., the dotted boxes in Figure 4.1) could be made interoperable by wrapping it up with a translator, which makes the database talk in the mediation language, and a conflict detector, which gives users the option of being notified of potential problems in query mediation. The users and applications accessing data in the legacy database become capable of accessing data in multiple databases without having to switch first to a new language or new schema.

Even though supporting the interoperation of legacy databases and legacy applications is the primary benefit of the mediation architecture in Figure 4.1, it does not exclude the incorporation of new databases or new schemas, when old schemas in legacy databases do not meet the need of a new application. A participating database could be a virtual one containing only a schema but no data, serving purely as an interface to autonomous heterogeneous databases (e.g., the one on the right in Figure 4.1). For example, an application designer could define his favorite schema, and specify some relationships of his schema with other participating databases. From then on, users of the application could formulate queries in this schema, and get meaningful access to related data in other databases through query mediation.

Although we assume that the mediator's knowledge is given, this knowledge does not have to be complete. The more knowledge the mediator has, the more data it could help communicate. In other words, a participating database does not have to be completely definable as a view on other participating databases. For example, suppose that two databases represent allergy and test data differently. If the mediator has knowledge about how the allergy data in two databases are related, then users could query one database and access allergy data in both databases through query mediation. If, on the other hand, the mediator does not have knowledge about how the test data in two databases are related, then users could not access test data in both databases by querying only one database.

We emphasize that our architecture accommodates the federation architecture [51] as a special case. For example, the virtual database on the right in Figure 4.1 could be considered as a federated schema. If the federated schema is constructed from the mediator's knowledge base by removing semantic and representational discrepancies and redundancies, and queries are mediated only in the direction from the federated schema to databases A and B, then we get a federated database in which all queries go through the federated schema to access data in both databases A and B, as shown in Figure 4.2 (arrows represent data flow). Comparing Figures 4.1 and 4.2, we observe that data could flow from DB B to Wrapper A in Figure 4.1, because queries to database B are mediated to database A to access related data in database A. Such data flow is not possible in Figure 4.2, because queries have to be issued to the federated interface in order to access data in both databases A and B.

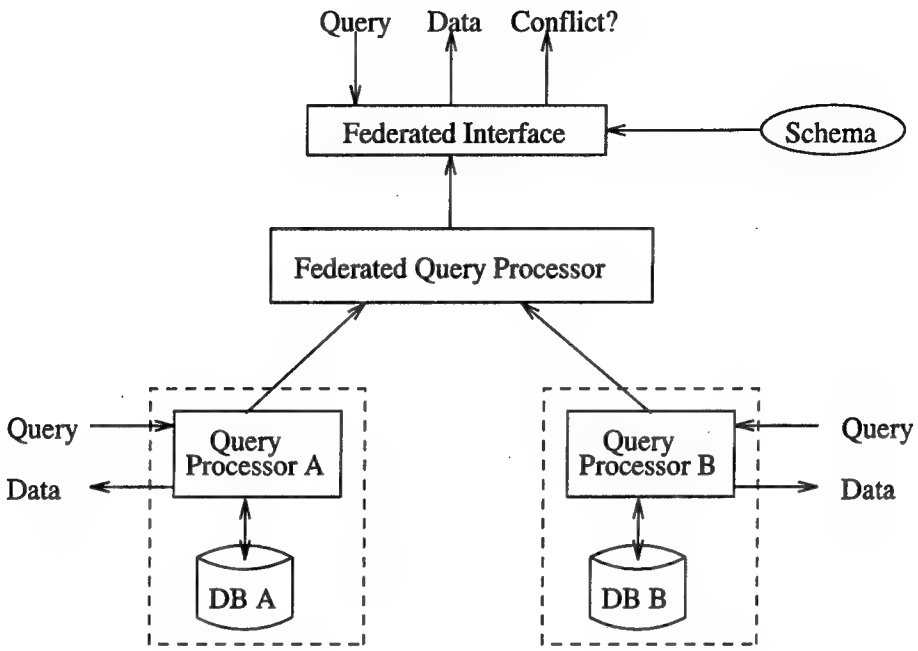


Figure 4.2: Federation Architecture

We also emphasize that the mediator's knowledge base, together with all the participating schemas, should not be equated to a global or federated schema, for the following reasons:

1. The mediator's knowledge base is at most a very poor schema, because it contains semantic and representational discrepancies and redundancies.

2. The semantic and representational discrepancies and redundancies in the mediator's knowledge base are not removed, since the removal would violate autonomy and would have high complexity.

3. The mediator's knowledge base is not the schema with which users interact.

Point (2) above shows a big advantage of our architecture over the federation architecture in terms of automation: users or database designers only need to identify, but do not have to resolve, the semantic and representational mismatches in order to access data in multiple databases, thus removing a big hurdle to automation.

Point (3) above shows another important advantage of our architecture over the federation architecture in terms of autonomy: users of a local database access data in multiple databases through the local language and schema instead of a federated schema or a multidatabase language. This is especially appealing for legacy databases: both the data and the applications accessing the data are interoperable.

In general, mediators are knowledge base systems. Since it is unrealistic to expect a single, general-purpose mediator with optimal power [5], multiple mediators should coexist (just like the

coexistence of multiple federated schemas in the federated database approach), offering *information communication services* at various levels [34, 60]. These mediators could differ in their trade-offs between communication cost and capability (bandwidth), and users would *subscribe* to the services that are optimal for their applications.

Chapter 5

Query Mediation

We choose first-order logic as our mediation language. As we can see in this chapter, query mediation becomes logical inference in first-order logic, and often in an even more efficient fragment of first-order logic, such as the fragment of Horn clauses. Here, we focus on the query transformer, since, for relational databases, the translation between relational query languages (e.g., SQL) and first-order logic is reasonably straightforward: see Chapter 6 for an example. The translation between object query languages (e.g., XSQL [24]) and first-order logic is discussed elsewhere [40]. In the rest of this chapter, schemas, databases, and queries are all formulated in our mediation language.

5.1 Schemas and Databases

Intuitively, a database represents a *perception* (called the *perceived world* [35] or the *model world* [48]) of the real world. Data in a database represents the knowledge of truth values of statements about the real world. A schema specifies the *vocabulary* in which data are expressed, and the *invariant* properties of data. It also supplies a context within which queries could be expressed meaningfully.

Formally, a *dependency* is a sentence in first-order logic of the form

$$(\forall x_1)(\forall x_2)\ldots(\forall x_m).\ p_1 \wedge p_2 \wedge \cdots \wedge p_k \supset (\exists y_1)(\exists y_2)\ldots(\exists y_n)(q_1 \wedge q_2 \wedge \cdots \wedge q_l)$$

where $m, n \geq 0$, p_i is an atomic formula for $1 \leq i \leq k$, and q_j is either an atomic formula or an equality (when $n = 0$ and $l = 1$) for $1 \leq j \leq l$. A dependency is *equality generating* if $n = 0$, $l = 1$, and q_1 is an equality. A dependency is *tuple generating* if q_j is an atomic formula for $1 \leq j \leq l$ [12].

A *schema* S is a theory (V, C) in first-order logic, where V is a vocabulary of predicate symbols called *relation schemes*, arguments of relation schemes are called *attributes*, and C is a set of equality-generating or tuple-generating dependencies expressed in V and called *integrity constraints*.

A *database* A over S is a structure over V, consisting of a nonempty domain D and, for every n-ary predicate symbol in V, an assignment of that predicate symbol to a mapping from its attributes into D. Database A is *valid* if it is a model of S. Given two databases A_1 and A_2 over schemas

$S_1 = (V_1, C_1)$ and $S_2 = (V_2, C_2)$, respectively, such that $V_1 \cap V_2 = \emptyset$, $A_1 \cup A_2$ is the database over $(V_1 \cup V_2, C_1 \cup C_2)$ that assigns the same value to every predicate P in V_i as A_i does, for $i \in \{1, 2\}$.

For our scenario in Chapter 2, the schema of the clinic database consists of three predicate symbols, Patients, Patient_Allergy, and Notes, together with integrity constraints such as

$$(\forall x)(\forall y)(\forall z).\ \text{Patients}(x, y) \wedge \text{Patients}(x, z) \supset y = z$$

$$(\forall x)(\forall y)(\forall z)(\forall u)(\forall \bar{v}).\ \text{Patient\_Allergy}(x, y, z, u, \bar{v}) \supset (\exists w)\ \text{Patients}(x, w)$$

A *conjunctive query* q on S is a conjunction of atomic formulas over V with a (possibly empty) list of free variables. A *logical query* q on S is a disjunction of conjunctive queries on S. Given a database A over S with domain D and a logical query q with free variables $x_1, x_2, \ldots, x_m$, the *answers* of q in A are the m-tuples $(d_1, d_2, \ldots, d_m)$ in D^m such that q is true in A when variables $x_1, x_2, \ldots, x_m$ are assigned the values $d_1, d_2, \ldots, d_m$, that is, such that $D \models q[d_1/x_1, d_2/x_2, \ldots, d_m/x_m]$. For our scenario in Chapter 2, the SQL query Q_C on the clinic database is equivalent to a conjunctive query, q_C:

$$(\exists y)(\exists z)(\exists w)(\exists \bar{v}).$$
$$\text{Patients}(x, y)$$
$$\wedge\ \text{Patient\_Allergy}(x, \text{xd2001}, z, w, \bar{v})$$
$$\wedge\ \text{Notes}(z, u)$$
$$\wedge\ 1994/01/01/08/00/00 < y).$$

We consider the *interoperation* of valid, autonomous, and heterogeneous databases A_i with domains D_i and over schemas $S_i = (V_i, C_i)$ for $1 \leq i \leq n$, where $V_i \cap V_j = \emptyset$ for $1 \leq i < j \leq n$. We assume that A_i is empty if the i-th database is virtual. The mediator's knowledge base consists of a theory $S = (\bigcup_{i=1}^{n} V_i \cup V, C)$ in first-order logic, where C is a set of tuple-generating dependencies. The mediator's knowledge captures the relationships among schemas $S_1, S_2, \ldots, S_n$, which specify how data in databases $A_1, A_2, \ldots, A_n$ are related semantically. For our scenario in Chapter 2, $V = \emptyset$, and the mediator's knowledge includes sentences such as the following:

$$\text{xd2001} = \text{experimental\_drug\_2001}$$

$$(\forall x)(\forall y)(\forall z).$$
$$\text{Drug\_Allergy}(x, y, z)$$
$$\supset (\exists u)(\exists v)(\exists \bar{w})\ (\text{Patient\_Allergy}(x, y, u, v, \bar{w}) \wedge \text{Notes}(u, z))$$

$$(\forall x)(\forall y)(\forall z)(\forall \bar{w})(\forall u).$$
$$\text{Admissions}(x, y, z, \bar{w}) \wedge u < y \supset \text{Patients}(x, y) \wedge u < y$$

The first formula means that the mediator knows that xd2001 and experimental_drug_2001 are the same, so that a reference to one can be replaced by a reference to the other without changing the meaning of a query. Similarly, the second formula makes an assertion about the relationship among meanings of the relations mentioned, not about the content of the databases. It means that every answer to the antecedent query

$$\text{Drug\_Allergy}(x, y, z)$$

should be a valid answer to the consequent query

$(\exists u)(\exists v)(\exists \bar{w})$. Patient_Allergy($x$, y, u, v, $\bar{w}$) $\wedge$ Notes(u, z).

In general, the relationships among schemas are not necessarily pairwise—there might be relationships involving three or more schemas (see Section 6.4 for an example).

5.2 Properties of Query Mediation

Consider the interoperation of databases A_i over schemas $S_i = (V_i, C_i)$ for $1 \leq i \leq n$. Suppose that the mediator's knowledge base consists of theory $S = (\bigcup_{i=1}^{n} V_i \cup V, C)$. Given a logical query q on S_1 with free variables $x_1, x_2, \ldots, x_m$, a *mediated query* p of q is a logical query on the combined schema $\bigcup_{i=1}^{n} S_i \cup S = (\bigcup_{i=1}^{n} V_i \cup V, \bigcup_{i=1}^{n} C_i \cup C)$ with the same list of free variables. Notice that, although q is expressed on one schema S_1, p could encompass multiple schemas from $S_1, S_2, \ldots, S_n$ and the mediator's knowledge base S (see Section 6.4 for an example).

A mediated query p is *sound* if it logically implies the original query using the mediator's knowledge. Intuitively, soundness means that every answer of the mediated query should be a valid answer of the original query. This is formally expressed as follows:

$$\bigcup_{i=1}^{n} C_i \cup C \models (\forall x_1)(\forall x_2)\ldots(\forall x_m).\ p \supset q.$$

Naturally, the disjunction of sound mediated queries is also a sound mediated query. A mediated query p is *trivial* if it is sound even when the mediator's knowledge base is empty, i.e, when $C = \emptyset$. Intuitively trivialness means that every answer of the mediated query is obtainable by asking the original query:

$$\bigcup_{i=1}^{n} C_i \models (\forall x_1)(\forall x_2)\ldots(\forall x_m).\ p \supset q.$$

A mediated query p is *complete* if it is logically implied by all possible mediated queries p' of q. Intuitively, completeness means that every valid answer of the original query is an answer of the mediated query:

$$\bigcup_{i=1}^{n} C_i \models (\forall x_1)(\forall x_2)\ldots(\forall x_m).\ p' \supset p$$

for every mediated query p' of q.

When the mediator's knowledge base is empty, any logical query q is the sound, trivial, and complete mediated query of itself. However, the sound, nontrivial, and complete mediated query of q does not exist. In general, if a sound, nontrivial, and complete mediated query exists, then it is always unique up to equivalence. That is, if p and p' are two sound, nontrivial, and complete mediated queries of q, then they are equivalent:

$$\bigcup_{i=1}^{n} C_i \models (\forall x_1)(\forall x_2)\ldots(\forall x_m).\ p \equiv p'.$$

5.3 Meaning of Query Mediation

Again, consider the interoperation of databases A_i over schemas $S_i = (V_i, C_i)$ for $1 \leq i \leq n$. Suppose that the mediator's knowledge base consists of theory $S = (\bigcup_{i=1}^{n} V_i \cup V, C)$. Given a

logical query q on S_1, the objective of query mediation is to replace the evaluation of q in A_1 by the evaluation of the sound, nontrivial, and complete mediated query p of q in the combined database $\bigcup_{i=1}^{n} A_i$. The soundness of p ensures that such replacement is meaningful with respect to the constraints in $S_1, S_2, \ldots, S_n$ and the relationships in S. For the scenario in Chapter 2, a mediated query q_H on the hospital database can be derived from logical query q_C and the mediator's knowledge in Section 5.1,

$(\exists y)(\exists z)(\exists \bar{w})$.
Admissions(x, y, z, $\bar{w}$)
$\wedge$ Drug_Allergy(x, experimental_drug_2001, u)
$\wedge$ 1994/01/01/08/00/00 $< y$

which ensures that the constant xd2001 is converted to experimental_drug_2001 before it is compared to DRUG_ID values, among other things. When translated to SQL, query q_H above becomes the SQL query Q_H of Chapter 2.

If the mediated query p is trivial, then the answers of p are contained in the answers of the original query q, since databases $A_1, A_2, \ldots, A_n$ are valid and

$$\bigcup_{i=1}^{n} C_i \models (\forall x_1)(\forall x_2)\ldots(\forall x_m).\ p \supset q.$$

Hence query mediation does not yield additional data. For the scenario in Chapter 2, suppose that we have an additional relation MEDICARE_PATIENTS recording those patients who are covered by Medicare. The fact that every Medicare patient is a patient can be captured by the integrity constraint

$(\forall x)$. Medicare_Patients(x) $\supset$ $(\exists y)$ Patients(x, y).

From query q_C and the above constraint, we could derive the following sound mediated query on the clinic database, which is a trivial one because its answers are contained in the answers of the original query.

$(\exists y)(\exists z)(\exists w)(\exists \bar{v})$.
Patients(x, y)
$\wedge$ Medicare_Patients(x)
$\wedge$ Patient_Allergy(x, xd2001, z, w, $\bar{v}$)
$\wedge$ Notes(z, u)
$\wedge$ 1994/01/01/08/00/00 $< y$.

The completeness of the mediated query p ensures that all the valid answers of the original query q, whether they are in databases $b_1, \ldots, b_n$, will be accessed by evaluating p. In the scenario of Chapter 2, the disjunction of queries q_C and q_H is a complete mediated query, ensuring that all patients who had recent allergic reactions to drug XD2001 are accessed, whether they are recorded in the clinic database or in the hospital database.

5.4 Semantics of Query Mediation

Consider, once again, the interoperation of databases A_i over schemas $S_i = (V_i, C_i)$ for $1 \leq i \leq n$. Suppose that the mediator's knowledge base consists of theory $S = (\bigcup_{i=1}^{n} V_i \cup V, C)$. Every equality-

generating dependency in $C_1, C_2, \ldots, C_n$ is a definite Horn clause. Through skolemization, every tuple-generating dependency in $C_1, C_2, \ldots, C_n$, or C of the form

$$(\forall x_1)(\forall x_2)\ldots(\forall x_m).\ p_1 \wedge \cdots \wedge p_k \supset (\exists y_1)(\exists y_2)\ldots(\exists y_n)(q_1 \wedge q_2 \wedge \cdots \wedge q_l)$$

could also be transformed into l definite Horn clauses:

$$q_1[f_i(x_1, x_2, \ldots, x_m)/y_i]_{1 \leq i \leq n} \leftarrow p_1, p_2, \ldots, p_k$$
$$\vdots$$
$$q_l[f_i(x_1, x_2, \ldots, x_m)/y_i]_{1 \leq i \leq n} \leftarrow p_1, p_2, \ldots, p_k$$

where $f_i(x_1, \ldots, x_m)$ is a Skolem function. For every m-ary predicate symbol P in $V_1, V_2, \ldots, V_n$, or V, we add a new m-ary predicate symbol P_0 and the following definite Horn clause:

$$P(x_1, x_2, \ldots, x_m) \leftarrow P_0(x_1, x_2, \ldots, x_m).$$

A deductive database (with equality) [13] could be constructed by taking these Horn clauses as the intensional database (IDB). The extensional database (EDB) consists of, for every predicate P in $V_1, V_2, \ldots, V_n$, or V, the new predicate P_0 whose extent is the relation assigned to P by $A_1, A_2, \ldots, A_n$, or A.

Let M be the initial model of this deductive database [28]. Also let D be the universe of M that is the set of equivalence classes of ground terms over $\bigcup_{i=1}^{n} V_i \cup V \cup \{f_1, \ldots, f_n\}$, and $G \subseteq D$ be the set of equivalence classes containing ground terms over $\bigcup_{i=1}^{n} V_i \cup V$. Given a logical query q on S_1 with free variables $x_1, x_2, \ldots, x_m$, the *definite answers* of q in M are the answers of q in M that are in G^m.

Given a mediated query p of q on the combined schema $\bigcup_{i=1}^{n} S_i \cup S$, if p is sound, then every answer of p in the combined database $\bigcup_{i=1}^{n} A_i \cup A$ is a definite answer of q in M. If p is trivial, then every answer of p in $\bigcup_{i=1}^{n} A_i \cup A$ is an answer of q in A_1 and hence, a definite answer of q in M. If p is complete, then every definite answer of q in M is an answer of p in $\bigcup_{i=1}^{n} A_i \cup A$.

When the IDB of this deductive database is bounded [57], there is a logical query p not involving IDB predicates, such that every answer of q in M is an answer of p in M and vice versa. Hence there is a logical query p' on the combined schema $\bigcup_{i=1}^{n} S_i \cup S$, such that every definite answer of q in M is an answer of p' in the combined database $\bigcup_{i=1}^{n} A_i \cup A$ and vice versa. In other words, p' is the sound and complete mediated query of q.

In general, we could view query mediation as the first-order approximation of definite answers in the initial model of a deductive database, which is formed by taking participating databases as the EDB, and by taking (the skolemization of) the mediator's knowledge and the constraints in participating schemas as the IDB. The more complete a mediated query is, the closer its answers are to the definite answers of the original query in the initial model. The boundedness of the IDB serves as a sufficient condition for the existence of sound and complete mediated queries.

Chapter 6

A Prototype Mediator

We have implemented a prototype mediator, which does SQL query mediation between Oracle databases. The system organization is shown in Figure 6.1, where the shaded box is to be implemented.

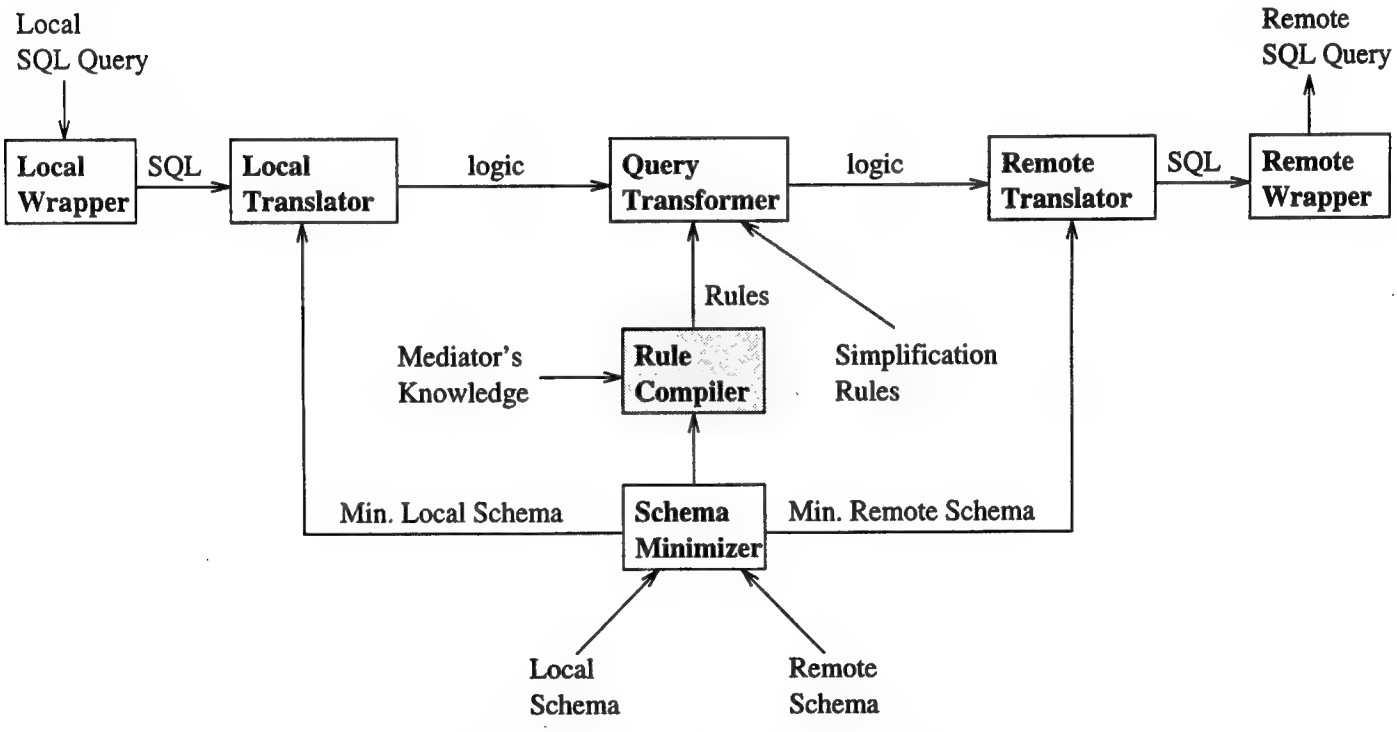


Figure 6.1: Prototype System Organization

We illustrate the details of our prototype implementation using the example query mediation of Chapter 2, from the SQL query Q_C on the clinic database A_C to the SQL query Q_H on the hospital database A_H.

6.1 Wrappers

The wrapper surrounding the local Oracle database intercepts SQL queries on the local database and sends them to the SQL-to-logic translator. The wrapper surrounding the remote Oracle database receives mediated SQL queries from the logic-to-SQL translator and sends them to the remote database.

6.2 Translators

We discuss the SQL-to-logic translator in detail. The logic-to-SQL translator is obtained by reversing the direction of the SQL-to-logic translation.

6.2.1 Step 1: Parsing

The first step performed by the SQL-to-logic translator is to parse an SQL query, which results in an abstract syntax tree. As part of the parsing process, omitted attribute qualifiers—such as the omitted NOTES qualifier on the attribute TEXT in query Q_C—are inferred and recorded.

6.2.2 Step 2: Representation Minimization

A schema can contain a great deal of representations that are artifacts of the chosen data model. Eliminating as many representations as possible reduces the potential representational mismatches of the local schema with the remote schema, making query mediation more efficient.

To minimize representations, we perform lossless decomposition of all relations in a schema as much as possible. When the integrity constraints in the schema are limited to key-based functional dependencies, representation minimization is farely straightforward. Suppose that relation R has attributes $X_1, X_2, \ldots, X_{m+n}$, where the first m of the $m+n$ attributes form the primary key. Let R_0 be the relation with attributes $X_1, X_2, \ldots, X_m$ that is obtained from R by projecting onto those key attributes, that is, $R_0 = \Pi_{X_1, X_2, \ldots, X_m} R$. Similarly, for $1 \leq i \leq n$, let $R_i = \Pi_{X_1, X_2, \ldots, X_m, X_{m+i}} R$. Then R is replaced by the set $\{R_0, R_1, \ldots, R_n\}$ in the minimized schema. The minimized schema also contains the following integrity constraints for $1 \leq i \leq n$:

$$(\forall x_1)(\forall x_2)\ldots(\forall x_m)(\forall y)(\forall z).\ R_i(x_1, x_2, \ldots, x_m, y) \wedge R_i(x_1, x_2, \ldots, x_m, z) \supset y = z$$

$$(\forall x_1)(\forall x_2)\ldots(\forall x_m)(\forall y).\ R_i(x_1, x_2, \ldots, x_m, y) \supset R_0(x_1, x_2, \ldots, x_m)$$

An SQL query on the original schema can be transformed into an SQL query on the minimized schema by replacing references to attribute X_i, where $1 \leq i \leq m$, of R by references to X_i of R_0 and by replacing references to attribute X_{m+i}, where $1 \leq i \leq n$, of R by references to X_{m+i} of R_i. So our original query Q_C is equivalent to query $\bar{Q}_C$

```
SELECT PATIENT_ALLERGY.PATIENT_ID, NOTES_TEXT.TEXT
FROM   PATIENT_ALLERGY,
       PATIENTS, PATIENTS_TRANSACTION_TIME,
       NOTES, NOTES_TEXT
```

```
WHERE  PATIENT_ALLERGY.PATIENT_ID = PATIENTS.PATIENT_ID
AND    PATIENT_ALLERGY.NOTE_ID = NOTES.NOTE_ID
AND    PATIENT_ALLERGY.DRUG_NAME = 'XD2001'
AND    TIMESTAMP '1994-01-01 08:00:00'
         < PATIENTS_TRANSACTION_TIME.TRANSACTION_TIME
AND    PATIENTS.PATIENT_ID = PATIENTS_TRANSACTION_TIME.PATIENT_ID
AND    NOTES.NOTE_ID = NOTES_TEXT.NOTE_ID;
```

on the minimized clinic schema. (The name "NOTES" is now used to refer to the projection of the original NOTES relation onto its primary key NOTES_ID, while "NOTES_TEXT" refers to the result of projecting NOTES onto its primary key NOTES_ID together with the attribute TEXT, and similarly for the other relations.)

6.2.3 Step 3: Translation to Logic

The last step of the translator replaces the abstract syntax tree for the minimized query by an equivalent logical query. Relations correspond to predicates; attributes correspond to variables[1]; selection conditions correspond to equalities and inequalities between variables; selected attributes correspond to free variables, and non-selected attributes correspond to existentially quantified variables. The minimized SQL query $\bar{Q}_C$ above is translated to logical query

(∃ ?pa.note_id)
(∃ ?pa.drug_name)
(∃ ?p.patient_id)
(∃ ?ptt.patient_id)
(∃ ?ptt.transaction_time)
(∃ ?n.note_id)
(∃ ?nt.note_id).
Patient_Allergy(?pa.patient_id, ?pa.drug_name, ?pa.note_id)
∧ Patients(?p.patient_id)
∧ Patients_Transaction_Time(?ptt.patient_id, ?ptt.transaction_time)
∧ Notes(?n.note_id)
∧ Notes_Text(?nt.note_id, ?nt.text)
∧ ?pa.patient_id = ?p.patient_id
∧ ?pa.note_id = ?n.note_id
∧ ?pa.drug_name = xd2001
∧ !1994/01/01/08/00/00 < ?ptt.transaction_time
∧ ?p.patient_id = ?ptt.patient_id
∧ ?n.note_id = ?nt.note_id

[1] We use qualified attribute names prefixed by '?' as variable names, to improve readability. However, we will abbreviate the qualifiers—writing, for example, "?p.patient_id" rather than "?patient.patient_id"—to reduce the length of argument lists. In our examples, no ambiguity results from such abbreviation.

6.3 Query Transformer

The query transformer derives a logical query on the remote database from a logical query on the local database. The result of the query transformer is passed on to the logic-to-SQL translator.

6.3.1 Step 1: Logical Simplification

When getting from the translator a logical query on some minimized schema, the query transformer first uses its knowledge of the integrity constraints in the minimized schema to simplify the query. Using the integrity constraints in Section 6.2.2, a logical query of the form

$$\begin{aligned} & R_i(x_1, x_2, \ldots, x_m, x_{m+i}) \\ & \quad \wedge R_0(y_1, y_2, \ldots, y_m) \\ & \quad \wedge x_1 = y_1 \\ & \quad \wedge x_2 = y_2 \\ & \quad \wedge \ldots \\ & \quad \wedge x_m = y_m \\ & \quad \wedge y_k < a \\ & \quad \wedge \ldots \end{aligned}$$

can be simplified to

$$\begin{aligned} & R_i(x_1, x_2, \ldots, x_m, x_{m+i}) \\ & \quad \wedge x_k < a \\ & \quad \wedge \ldots \end{aligned}$$

The query transformer automatically generates all such rules needed to eliminate logical redundancy introduced by minimization. In addition, the maintainer of the mediator's knowledge base can add simplification rules to eliminate any logical redundancy in the original schema. After simplification, the logical query of Section 6.2.3 becomes $\bar{q}_C$,

```
(∃ ?pa.note_id)
(∃ ?pa.drug_name)
(∃ ?ptt.patient_id)
(∃ ?ptt.transaction_time)
(∃ ?nt.note_id).
    Patient_Allergy(?pa.patient_id, ?pa.drug_name, ?pa.note_id)
            ∧ Notes_Text(?nt.note_id, ?nt.text)
            ∧ Patients_Transaction_Time(?ptt.patient_id, ?ptt.transaction_time)
            ∧ ?pa.patient_id = ?ptt.patient_id
            ∧ ?pa.note_id = ?nt.note_id
            ∧ ?pa.drug_name = xd2001
            ∧ !1994/1/1/8/0/0/0 < ?ptt.transaction_time
```

6.3.2 Step 2: Representation Transformation

We come now to the heart of the mediator, the step where the (minimized) representation used by the clinic database A_C is replaced by the representation used by the hospital database A_H. The

informal semantic relationships we noted between the schema of A_C and the schema of A_H, which were formalized in Section 5.1 in terms of equality and logical implication, are implemented in the prototype mediator's knowledge base by rewrite rules. The three rules relevant to our example are

$$\text{Patients}(p) \longrightarrow \text{Admissions}(p, ?\text{a.admission\_time})$$

$$\text{Patient\_Allergy}(p, d, n_1) \wedge \text{Notes\_Text}(n_2, x) \wedge n_1 = n_2 \longrightarrow \text{Drug\_Allergy\_Text}(p, d, x)$$

$$\text{Patients\_Transaction\_Time}(p, t_1) \wedge t_2 < t_1 \longrightarrow \text{Admissions}(p, t_1) \wedge t_2 < t_1$$

In addition, the query transformer has rules for changing from the terminology used in A_C to that used in A_H. The rule relevant to our example is

$$\text{xd2001} \longrightarrow \text{experimental\_drug\_2001}$$

Note that there are no rules for translating patient ids in A_C to patient ids in A_H. Unless there is reason to believe that patients are consistently identified across databases—the clinic and hospital might both use a patient's SSN as the id—logical queries containing particular patient ids cannot be rewritten from one representation to the other: a logical query such as

Patients(md919c)

where md919c is a particular patient id, can be rewitten to

(∃ ?a.admission_time). Admissions(md919c, ?a.admission_time)

using the first rule, but the mediator recognizes that this is not a proper query over A_H because md919c is not necessarily a term of the language for A_H.

Applying these rules to the simplified logical query $\bar{q}_C$ above, which defines a set of answers from A_C, produces a logical query $\bar{q}_H$ that defines a set of answers from A_H:

```
(∃ ?dat.drug_id)
(∃ ?a.patient_id)
(∃ ?a.admission_time).
    Drug_Allergy_Text(?dat.patient_id, ?dat.drug_id, ?dat.text)
            ∧ Admissions(?a.patient_id, ?a.admission_time)
            ∧ ?dat.patient_id = ?a.patient_id
            ∧ ?dat.drug_id = experimental_drug_2001
            ∧ !1994/1/1/8/0/0/0 < ?a.admission_time
```

where we have renamed some of the bound variables in $\bar{q}_H$ to make the structure clearer. Finally, the logic-to-SQL translator translates logical query $\bar{q}_H$ to an SQL query $\bar{Q}_H$ on the minimized hospital schema, which is, in turn, transformed into Q_H by deminimization.

6.4 Multiple Databases

Suppose that rather than a single database A_{H}, the hospital stores its information in several databases. In particular, suppose that the relation ADMISSIONS is part of database $A_{\mathrm{H}}^{(1)}$ and that the relation DRUG_ALLERGY is part of a different database $A_{\mathrm{H}}^{(2)}$. In that case, the rules for rewriting from A_{C} to $A_{\mathrm{H}}^{(1)}$ could include

$$\text{Patients}(p) \longrightarrow \text{Admissions}(p, ?\text{a.admission\_time})$$

$$\text{Patients\_Transaction\_Time}(p, t_1) \wedge t_2 < t_1 \longrightarrow \text{Admissions}(p, t_1) \wedge t_2 < t_1$$

while the rules for rewriting from A_{C} to $A_{\mathrm{H}}^{(2)}$ could include

$$\text{Patients}(p) \longrightarrow \text{Drug\_Allergy}(p, ?\text{drug\_allergy.drug\_id})$$

$$\text{Patient\_Allergy}(p, d, n_1) \wedge \text{Notes\_Text}(n_2, x) \wedge n_1 = n_2 \longrightarrow \text{Drug\_Allergy\_Text}(p, d, x)$$

An attempt to rewrite logical query $\bar{q}_{\mathrm{C}}$ of Section 6.3.1 into a logical query over $A_{\mathrm{H}}^{(1)}$ would result in

```
(∃ ?pa.drug_name)
(∃ ?nt.note_id)
(∃ ?a.patient_id)
(∃ ?a.admission_time).
    Patient_Allergy(?pa.patient_id, ?pa.drug_name, ?pa.note_id)
        ∧ Notes_Text(?nt.note_id, ?nt.text)
        ∧ Admissions(?a.patient_id, ?a.admission_time)
        ∧ ?pa.note_id = ?nt.note_id
        ∧ ?pa.patient_id = ?a.patient_id
        ∧ ?pa.drug_name = xd2001
        ∧ !1994/1/1/8/0/0/0 < ?a.admission_time
```

This logical query does not define a set of answers from $A_{\mathrm{H}}^{(1)}$, because the relations PATIENT_ALLERGY and NOTES_TEXT of the minimized schema of A_{C} are still mentioned. Similarly, an attempt to rewrite $\bar{q}_{\mathrm{C}}$ into a logical query over $A_{\mathrm{H}}^{(2)}$ would result in

```
(∃ ?dat.drug_id)
(∃ ?ptt.patient_id)
(∃ ?ptt.transaction_time).
    Drug_Allergy_Text(?dat.patient_id, ?dat.drug_id, ?dat.text)
        ∧ Patients_Transaction_Time(?ptt.patient_id, ?ptt.transaction_time)
        ∧ ?dat.patient_id = ?ptt.patient_id
        ∧ ?dat.drug_id = experimental_drug_2001
        ∧ !1994/1/1/8/0/0/0 < ?ptt.transaction_time
```

which does not define a set of $A_{\mathrm{H}}^{(2)}$ answers due to the mentioning of the (minimized version of) A_{C} relation PATIENTS_TRANSACTION_TIME.

But there is a clear sense in which the first rewriting attempt tells us that $A_H^{(1)}$ provides good *partial* information, and we might decide to pursue the matter further. If we then attempted to apply the rules for rewriting from A_C to $A_H^{(2)}$ to that logical query, we obtain $\bar{q}_H$. If we have reason to believe that $A_H^{(1)}$ and $A_H^{(2)}$ are using consistent patient ids, so that the comparison

?dat.patient_id = ?a.patient_id

makes sense—more generally, if we have a method for converting from $A_H^{(1)}$ patient ids to $A_H^{(2)}$ patient ids as part of the rule set for rewriting from $A_H^{(1)}$ to $A_H^{(2)}$—then $\bar{q}_H$ is easily transformed into a query on $A_H^{(1)}$, a query on $A_H^{(2)}$, and some "glue" code that the mediator can use to join the sets of answers to those two queries into a set of answers to $\bar{q}_C$. By using this technique, the mediator can combine information from multiple databases in responding to a query.

Not every relation among multiple databases can be broken down into relations between pairs of databases in the fashion illustrated in our example. Suppose that, instead of using consistent patient ids, $A_H^{(1)}$ and $A_H^{(2)}$ use different ids; say, $A_H^{(2)}$ uses the patient's SSN rather than the arbitrary unique id used in $A_H^{(1)}$. If the information for converting between $A_H^{(1)}$ ids and $A_H^{(2)}$ ids is stored, for confidentiality reasons, in the relation PATIENT_DATA of yet a third database $A_H^{(3)}$, then the rules for rewriting patient ids from $A_H^{(1)}$ to $A_H^{(2)}$ and conversely must contain references to that relation. Thus, the rules can no longer be thought of as simply relating $A_H^{(1)}$ and $A_H^{(2)}$. The rule for rewriting patient ids from $A_H^{(1)}$ to $A_H^{(2)}$ still has the general form

$$\langle \textit{pattern that matches } A_H^{(1)} \textit{ patient ids} \rangle \longrightarrow \langle \textit{some function of that pattern that produces } A_H^{(2)} \textit{ patient ids} \rangle$$

but the function now involves generating and executing a query on $A_H^{(3)}$, rather than simply performing a syntactic transformation of the left hand side. The result of looking up the patient's SSN in $A_H^{(3)}$ is used in rewriting the query on $A_H^{(1)}$ to a query on $A_H^{(2)}$. Fortunately, given the combinatorics, such situations are rare. The point is simply that effective query mediation can require knowledge of semantic relationships among several different databases, not just between pairs of databases.[2]

6.5 Prototype Implementation

The mediator is written in Common Lisp, and the wrappers are written in C. Communications between the wrappers and the translators are implemented by a combination of low-bandwidth IPCs used as signals and text files that contain the queries and answers.

The details of the prototype system design and implementation, as well as a set of demonstration scenarios, are presented in the Appendix.

[2]The federated database approach would translate all data values used by the databases into a single privileged representation that is regarded as expressing the semantics of the others, for example, universal patient ids. The problem with this approach is that it is much harder to design and maintain such a comprehensive, all-purpose representation than it is to specify the semantic relationships between some given databases of interest.

Chapter 7

Related Work

The dominating approach to the interoperation of heterogeneous databases has been that of the federated database [1, 51]. As we observed in Chapter 3, users and applications of a local database must switch to a federated schema or a multidatabase language to access data in multiple databases, which almost always involve different data models and query languages. For example, SIMS [1] requires users and applications to access multiple data sources using the Loom knowledge representation language and a domain model encoded in the Loom knowledge base.[1] In other words, the data in a local database are made interoperable, but the applications that access the data in the local database remain not interoperable, since these applications are coded in the data model and query language of the local database. This is especially impractical for legacy databases because the bulk of the significant investment made by organizations in such databases is in the applications that access the data. In contrast, both data and applications are made interoperable with our mediator approach.

In the federated database approach, either a federated database administrator or a user must first *identify* the semantic and representational mismatches, and then construct a federated schema to *resolve* these mismatches, before data in multiple databases could be accessed. The construction of the federated schema is essentially a schema integration process [3], which offers little hope for automation [49]. In comparison, our mediator approach does not need the mismatches to be resolved and removed in the form of an integrated schema.

In addition, most researchers advocate the use of a powerful interoperation language in federated databases that could directly express all the representational constructs of heterogeneous databases [2, 9, 23, 25]. Although mapping heterogeneous databases into constructs of the language becomes straightforward, all the semantic and representational mismatches still have to be resolved in the language, which offers little hope for efficiency because of the rich semantics and representations of the language. For example, higher-order logic must be used in [14] to reason about the equivalence of heterogeneous representations. In contrast, we advocate using first-order logic as our mediation

[1] Theoretically it is possible, in the reference architecture of [51], to have external schemas whose data models and query languages are different from a federated schema. However, it remains open whether and how this could be done with the federated database approach. Moreover, having an external schema identical to a local schema would introduce architectural redundancy.

language, which is more efficient than higher-order logics. The use of representation minimization techniques further improves the efficiency of query mediation.

The idea of information processing and communication via intelligent mediation is introduced in [60] as a framework of future information systems. Meta-attributes have been used in [47] to specify the contexts associated with attribute values. Relationships between contexts are encoded as conversion rules, and attribute values are mediated through these relationships to ensure that they are meaningful with respect to their contexts. This is a special case of query mediation, where the mediation is restricted to context matching and value conversion. An example of query mediation from object-oriented databases to relational databases is given in [42], where the schemas and the relationships between schemas are encoded as rules in F-logic. We support query mediation in its full generality, including the mediation of high-order object queries to first-order relational queries [40]. Our mediation language is more efficient than F-logic, making the correctness of query mediation much easier to define and verify.

It is first observed in [59] that data should be shared in some mediation language with minimal representational bias. There, the relational model is proposed as such a language, from which object-oriented views are compiled by binding relational data to object templates. The relational model has been used as the mediation language for resolving domain mismatches [10] and as the glue language for interconnecting software components [4]. In [15], first-order logic is recommended as the language for knowledge sharing. Our mediation language is essentially the language of the relational model, and our representation minimization techniques further reduce the representational bias.

Chapter 8

Conclusion

We have presented a query mediation approach to the interoperation of autonomous heterogeneous databases containing data with semantic and representational mismatches. We have developed a mediation architecture of interoperation that facilitates query mediation, and have formalized the semantics of query mediation. Queries are mediated between multiple databases, and users and applications of a local database access data in multiple databases using the local language and schema, making both the data and the applications accessing the data in legacy databases interoperable. Queries are automatically mediated, relieving users from the difficult task of resolving semantic and representational mismatches. Semantic heterogeneity is separated from representational heterogeneity by representation minimization techniques, reducing the space of heterogeneity and improving the efficiency of automated query mediation. Our approach provides a seamless migration path for legacy databases, enabling organizations to leverage off investments in legacy data and legacy applications.

Much research remains to be done. First, we have focused on three components of the mediation architecture, namely, the mediation language, the query transformer, and the translator for object query languages [40]. Research is needed in the other components as well as in translators for other kinds of query languages.

Second, we have assumed that the knowledge in the mediator's knowledge base is available. How to obtain such knowledge is certainly an important issue. Although the acquisition of such knowledge is likely to be a highly interactive process, automated acquisition tools would be valuable.

Third, we have restricted ourselves to constraints, relationships, and queries that do not involve negation. The semantics of query mediation could certainly be generalized to allow negation, as long as, for example, the result is stratified [57]. The approach could also be easily generalized to the interoperation with deductive databases containing rules in addition to constraints.

Finally, research is needed in the autonomous optimization of mediated query evaluation. Because of the autonomy of participating databases, the mediator often does not have access to the performance information that is crucial in query optimization. The mediator needs a cost model that is independent of the implementation structures of participating databases [61]. Techniques are also needed for the mediator to obtain performance information by querying [11].

Part II

A MAC Policy Framework

Chapter 9

Introduction

As more multilevel databases are built and connected through computer networks, a wide variety of secure data sources will become accessible. A big challenge presented by this technology is the secure interoperation of multilevel databases containing data with mismatched security policies. Providing secure interoperation of multilevel databases not only makes it possible to reliably share data in isolated military and civilian databases, but also increases users' confidence and willingness in such sharing.

9.1 Problem

As a prerequisite to the secure interoperation of multilevel databases containing data with mismatched security policies, the security policies of component databases, as well as the potential mismatches between them, have to be precisely characterized. Existing literature has been vague on what constitutes a security policy, its content ranging from high-level specifications such as the type of access control (mandatory or discretionary access control) or the kind of model (noninterference or Bell-LaPadula), to designer's belief or preferences such as whether polyinstantiation is allowed, to low-level specifications such as the number of levels and categories allowed in a lattice. A formal policy framework is needed within which security policies could be characterized and compared [20].

It has been widely accepted that a mandatory access control (MAC) policy consists of four components: a set of subjects, a set of objects, a lattice, and a mapping that associates levels in the lattice to subjects and objects [27]. This works well for multilevel operating systems, because objects such as files do not carry semantics. For multilevel databases where data carry semantics, the same mapping of levels to objects such as elements in tuples could have completely different meanings [52]. For example, consider a relation SMD(Starship, MId, Destination). A secret label on element Rigel of tuple (Enterprise, 101, Rigel) in SMD could mean that the fact "Enterprise is going to Rigel" is secret, or the fact "some starships are going to Rigel" is secret, or even the word "Rigel" is secret. This confusion suggests that something critical is missing with the traditional formulation of MAC policies in multilevel databases, namely the semantics of object labels. This problem is crucial in the secure interoperation of multilevel databases. For example, if the secret

label on Rigel means that the fact "some starships are going to Rigel" is secret in database A, and means that the word "Rigel" is secret in database B, then unclassified users could query all existing destinations in database A and obtain "Rigel" through interoperation with database B. The canonical MAC policy for federated databases proposed in [36] does not solve this problem.

The formulation of a MAC policy in a multilevel database often includes some constraint policies, such as the labeling policy of Seaview [30] and the classification constraints of LDV [18]. Constraints are the most important means of specifying data semantics. However, existing multilevel databases provide neither a precise definition of constraint validity nor an efficient mechanism of constraint enforcement. In fact, it has been argued [8, 32] that integrity enforcement is in fundamental conflict with secrecy enforcement: no multilevel databases could simultaneously satisfy both integrity and secrecy requirements.

An important characteristic of MAC policies is the upward information flow in the lattice, which indicates the believability of low data at high levels. For multilevel operating systems where objects do not carry semantics, low data are always believed at high. For multilevel databases where data carry semantics expressed by constraints however, low data could contradict high data. For example, if we require that high SMD tuples have unique MId elements and (Enterprise, 101, Rigel) is a high tuple in SMD, then the low tuple (Enterprise, 102, Rigel) in SMD could not be believed at high. This problem suggests that upward information flow should be constrained in the formulation of MAC policies in multilevel databases.

Constraints also bring about the danger of inference channels. Inference channels could be obtained either by knowing the constraints enforced by a database or by observing the behavior of a database in enforcing the constraints. For example, consider another relation MT(MissionId, Type). If we require that every MId element in relation SMD refers to a MissionId element in MT, and a low MId element refers to a high MissionId element, then low users could infer the existence of the high MissionId element. If we require that every high MId element in SMD refers to a low MissionId element in MT, then the attempt to delete a low MissionId element referred to by a high MId element would either cause a loss of high data or enable low users to infer the existence of the high MId element. Thus the formulation of MAC policies in multilevel databases should provide additional means to detect and remove such inference channels.

9.2 Overview of This Part

We have developed a formal policy framework for MAC policies in multilevel relational databases, which serves as the basis for specifying such policies and for characterizing their potential mismatches. In Chapter 10, we describe our framework and identify the components of MAC policies. In Chapter 11 we introduce the (single-level) relational model and the notion of atomic decomposition, which will be used repeatedly in the following chapters. In Chapters 12 through 14, we investigate in detail the three most important components of our policy framework.

In particular, Chapter 12 presents interpretation policies which map multilevel relational databases with tuple-level and element-level labeling to logical theories and structures. Chapter 13 presents view policies as means to constrain upward information flow in the lattice, identifies desirable properties of such policies, and develops a view policy that satisfies these properties.

Chapter 14 presents update policies as means to enforce constraints without introducing inference channels, identifies desirable properties of such policies, and develops an update policy that satisfies these properties.

Our framework could be used to capture and resolve the MAC policy mismatches in the secure interoperation of heterogeneous multilevel databases. In Chapter 15, we take an initial step in this direction, by investigating the secure interoperation of multilevel databases whose MAC policies mismatch in one specific component—the lattice. Finally, Chapter 16 offers some concluding remarks and a brief discussion of future work.

Due to space limitations, formal proofs of the results presented in Chapters 11 through 15 are not included. Interested readers can find them in [17, 37, 39, 41].

Chapter 10

A Policy Framework

In this chapter, we first develop a logical foundation of multilevel relational databases. We then present a framework of MAC policies based on the logical foundation, and identify the components of our framework.

10.1 A Model-Theoretic Formulation of Multilevel Relational Databases

A *state of the world* could be envisioned as a set of elements linked together by relationships. Information in a state of the world is the knowledge of the truth value of a statement about the state of the world [35], which could be either an elementary fact such as "Enterprise is on mission #101 to Rigel" or a general law such as "starships have unique missions".

A *relational database* captures a finite set of elements linked together by relationships. Elementary facts are represented as tuples in relations, and general laws are represented as integrity constraints. For example, the elementary fact "Enterprise is on mission #101 to Rigel" could be represented by the tuple (Enterprise, 101, Rigel) in relation SMD, and the general law "starships have unique missions" is represented by a functional dependency SMD: Starship $\rightarrow$ MId.

A standard model-theoretic formulation of a (single-level) relational database is to interpret the integrity constraints as forming a first-order theory, and the relations as forming a first-order structure of the theory [35]. A database is valid if the structure is a model of the theory. For example, the tuple (Enterprise, 101, Rigel) is interpreted as a tuple in the assignment to predicate SMD, and the functional dependency SMD: Starship $\rightarrow$ MId is interpreted as the axiom

$$(\forall x, y_1, y_2, z_1, z_2)(\text{SMD}(x, y_1, z_1) \wedge \text{SMD}(x, y_2, z_2) \rightarrow y_1 = y_2).$$

A *multilevel state of the world* is a family of states of the world, one for every level in a security lattice. Information in a multilevel state of the world is the knowledge of the truth value of a statement about a state of the world [55] or about the multilevel state of the world. The former could be either a classified elementary fact such as "it is top-secret that Enterprise is on mission #101 to Rigel", or a classified general law such as "it is confidential that starships have unique

missions". The latter could be a general law on classification such as "starships classified at all levels have unique missions".

A *multilevel relational database* is a relational database, whose integrity constraints are called *view constraints*, together with a *labeling function* κ and a set of *labeling constraints*. The labeling function maps every object in the database — relation, attribute, tuple, element in a tuple, view constraint, etc. — to a (possibly empty) set of levels in a security lattice. For a multilevel state of the world, the database and the labeling function together represent the family of states of the world, and the labeling constraints represent the general laws on classification[1]. For example, the tuple (Enterprise, 101, Rigel) mapped to ts by κ represents the elementary fact "it is top-secret that Enterprise is on mission #101 to Rigel". As a view constraint, the functional dependency SMD: Starship $\rightarrow$ MId mapped to c by κ represents the general law "it is confidential that starships have unique missions". As a labeling constraint, the functional dependency SMD: Starship $\rightarrow$ MId represents the general law "starships classified at all levels have unique missions".

The above observation suggests a model-theoretic formulation of multilevel relational databases as follows. A *multilevel theory* is a triplet $(\mathcal{L}, \{T^l\}_{l \in L}, C)$:

1. $\mathcal{L} = (L, \preceq)$ is a security lattice where L is a set of *levels* and $\preceq$ is the *dominance relation*,
2. $\{T^l\}_{l \in L}$ is a family of first-order theories — one for every level in L, each of which representing the view constraints classified at a particular level, and
3. C is a collection of axioms representing the labeling constraints.

We use $\prec$ to denote the strict dominance subrelation, and $\preceq^*$ to denote the transitive closure of $\preceq$. A *multilevel structure* of the multilevel theory is a family of first-order structures $\{M^l\}_{l \in L}$ where M^l is a structure of theory T^l.

For example, the tuple (Enterprise, 101, Rigel) mapped to ts is interpreted as a tuple in the assignment to predicate SMD^{ts} in structure M^{ts}, the view constraint SMD: Starship $\rightarrow$ MId mapped to c is interpreted as the axiom

$$(\forall x, y_1, y_2, z_1, z_2)(\text{SMD}^{\text{c}}(x, y_1, z_1) \wedge \text{SMD}^{\text{c}}(x, y_2, z_2) \rightarrow y_1 = y_2)$$

in theory T^{c}, and the labeling constraint SMD: Starship $\rightarrow$ MId is interpreted as the axiom

$$(\forall l_1, l_2 \in \mathcal{L})(\forall x, y_1, y_2, z_1, z_2)(\text{SMD}^{l_1}(x, y_1, z_1) \wedge \text{SMD}^{l_2}(x, y_2, z_2) \rightarrow y_1 = y_2)$$

in C.

10.2 MAC Policy

We restrict ourselves to multilevel relational databases whose MAC policies have the simple security property and the $*$-property of the Bell-LaPadula model [27], which ensure that information does not flow downward in the lattice.

[1] Since labeling constraints themselves are not objects in the database, they are not labeled by the labeling function.

- **The Simple Security Property.** A process is allowed a read access to a tuple only if the former's clearance level is identical to or higher than the latter's classification level in the lattice.
- **The $*$-Property.** A process is allowed a write access to a tuple only if the former's clearance level is identical to or lower than the latter's classification level in the lattice.

Our formulation of a MAC policy in a multilevel relational database has seven components:

1. a lattice,
2. a set of subjects,
3. a set of objects,
4. a mapping of subjects and objects to levels in the lattice,
5. an interpretation policy,
6. a view policy, and
7. an update policy.

The first four components together correspond to the traditional formulation of MAC policies in multilevel operating systems.

An interpretation policy maps a multilevel relational database to a multilevel theory and a multilevel structure of the theory. Through this policy, the superficial syntactic difference in object labels is abstracted away, and the semantic difference hidden in object labels is made precise. As a consequence, the interpretation policy makes it possible to compare the semantics of multiple MAC policies.

A view policy consists of a set of view constraints and a specification of the validity of view constraints. This policy specifies the upward information flow requirements.

An update policy consists of a set of labeling constraints, a set of updates, and a specification of the enforcement of labeling constraints in performing the updates. This policy specifies the mechanisms to eliminate inference channels in the enforcement of labeling constraints.

In the rest of this report, we develop the last three components of our policy framework, using examples from multilevel relational databases based on the lattice in Figure 10.1.

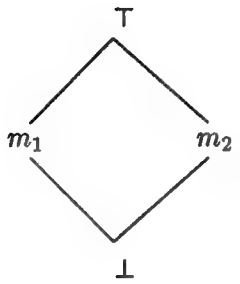


Figure 10.1: A Lattice

Chapter 11

Relational Model

Before defining the multilevel relational model, we need to define the (single-level) relational model. Following the practice of most existing approaches, we consider the relational model [56] extended with two important classes of constraints: key-based functional and referential dependencies. We then develop the technique of atomic decomposition, and characterize the information content of relational databases using the technique. The results obtained here will be used repeatedly in the following chapters.

11.1 Basic Notations

Let U be a finite set of *attributes*. If X, Y are subsets of U, then XY denotes the union of X, Y. If $A \in U$, then XA denotes $X\{A\}$. A *relation scheme* (in Boyce-Codd Normal Form) $R[X, K]$ is a set of attributes $X \subseteq U$ named R with nonempty *primary key* $K \subseteq X$. A *database schema* is a pair $(\mathcal{R}, \mathcal{C})$, where $\mathcal{R} = \{R_i[X_i, K_i]\}_{1 \leq i \leq n}$ is a family of relation schemes and $\mathcal{C}$ is a set of *key-based referential dependencies*:

1.1 Every referential dependency in $\mathcal{C}$ has the form $R_i[Y] \hookrightarrow R_j$, where $1 \leq i, j \leq n$, $Y = K_i$ or $Y \subseteq X - K_i$, and $|Y| = |K_j|$. Y is a *foreign key* in relation scheme R_i to relation scheme R_j.

1.2 Distinct foreign keys in the same relation scheme are disjoint. In other words, $Y = Z$ or $Y \cap Z = \emptyset$ for $1 \leq i, j, k \leq n$ and $R_i[Y] \hookrightarrow R_j, R_i[Z] \hookrightarrow R_k$ in $\mathcal{C}$.

For relation scheme $R_i[X_i, K_i]$ in $\mathcal{R}$ and attribute $A \in X_i$, A is a *nonkey* attribute if $A \notin K_i$ and $A \notin Y$ for any foreign key Y in R_i. Figure 11.1 shows a schema with two relation schemes SMD and MT, where boxes represent relation schemes, attributes to the left of double lines form primary keys, and arrows between boxes represent referential dependencies.

Let $\mathcal{D}$ be a (possibly infinite) set of values. A *tuple* over attributes X is a partial mapping $t[X]: X \mapsto \mathcal{D}$ that assigns values from $\mathcal{D}$ to attributes in X. For attribute $A \in X$, $t[A]$ denotes the value assigned to A by $t[X]$, and $t[A] = \angle$ denotes that $t[A]$ is undefined[1]. For attributes $Y \subseteq X$,

[1]We distinguish between ***unknown*** nulls and ***not-applicable*** nulls. The symbol $\angle$ represents unknown nulls. Unknown nulls say that some elementary facts are missing from the database. Because a database is not a part of the

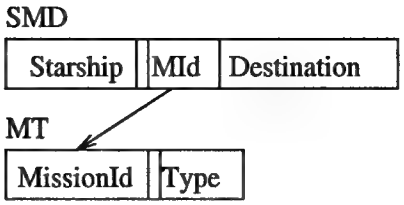


Figure 11.1: A Schema

$t[Y]$ denotes the partial mapping whose domain is restricted to attributes in Y. For tuple t over X, $t[X] = \angle$ denotes that t is empty: $t[A] = \angle$ for all attributes $A \in X$; and $t[X] \neq \angle$ denotes that t is total: $t[A] \neq \angle$ for all attributes $A \in X$.

A *relation* r over relation scheme $R[X, K]$ is a set of tuples over X. For attributes $Y \subseteq X$, $r[Y]$ denotes the set of tuples $t[Y]$ where $t \in r$. Relation r is *valid* if it satisfies the *key integrity property*:

2.1 for every tuple $t \in r$, $t[K] \neq \angle$, and

2.2 for every pair of tuples $t, t' \in r$, $t[K] = t'[K]$ implies $t = t'$.

In other words, tuples with the same primary key value are identical.

A *database* b over database schema $(\mathcal{R}, \mathcal{C})$, where $\mathcal{R} = \{R_i[X_i, K_i]\}_{1 \leq i \leq n}$, is a family of relations $\{r_i\}_{1 \leq i \leq n}$, where r_i is a relation over $R_i[X_i, K_i]$. It is *r-valid* if every relation in b is valid. It is *valid* if it is r-valid and satisfies the *referential integrity property* for every referential dependency $R_i[Y] \hookrightarrow R_j$ in $\mathcal{C}$ and tuple $t \in r_i$:

3.1 either $t[Y] = \angle$ or $t[Y] \neq \angle$, and

3.2 if $t[Y] \neq \angle$ then there is a tuple $t' \in r_j$ such that $t[Y] = t'[K_j]$.

In other words, every non-null foreign key value refers to an existing primary key value. $\mathcal{D}$ is the *universe* of b. Below is a valid database over the schema of Figure 11.1.

Starship	Mission	Destination
Enterprise	101	Rigel
Voyager	102	Talos
Discovery	103	Rigel

MissionId	Type
101	spy
102	explore
103	mine

For $Y \subseteq X$, the *total projection* of relation $r[X]$ to Y, denoted as $\Pi_Y r[X]$, is defined as the set of tuples $t[Y]$ such that $t[Y] \in r[Y]$ and $t[Y] \neq \angle$.

state of the world that the database tries to capture, unknown nulls do not represent elementary facts in the state of the world.

11.2 Atomic Decomposition

Every tuple in a database represents an elementary fact. Often, the elementary fact represented by a tuple is a conjunction of several *smaller* elementary facts. For example, tuple (Enterprise, 101, Rigel) represents the elementary fact "Enterprise is on mission #101 to Rigel", which is the conjunction of two smaller elementary facts "Enterprise is on mission #101" and "Enterprise goes to Rigel".

Let $\mathcal{B} = (\mathcal{R}, \mathcal{C})$ be a schema where $\mathcal{R} = \{R_i[X_i, K_i]\}_{1 \leq i \leq n}$. The *atomic decomposition* of $\mathcal{B}$ is a schema $\mathcal{B}^a$ consisting of the following set of relation schemes $\mathcal{R}^a$:

- $R_i^K[K_i, K_i]$ for every $R_i[X_i, K_i]$ in $\mathcal{R}$,
- $R_i^Y[K_iY, K_i]$ for every $R_i[X_i, K_i]$ in $\mathcal{R}$ and foreign key Y in R_i where $Y \subseteq X_i - K_i$, and
- $R_i^A[K_iA, K_i]$ for every $R_i[X_i, K_i]$ in $\mathcal{R}$ and nonkey attribute $A \in X_i - K_i$;

and the following set of key-based referential dependencies $\mathcal{C}^a$:

- $R_i^Y[K_i] \hookrightarrow R_i^K$ and $R_i^A[K_i] \hookrightarrow R_i^K$ for every $R_i^K[K_i, K_i]$, $R_i^Y[K_iY, K_i]$, and $R_i^A[K_iA, K_i]$,
- $R_i^K[K_i] \hookrightarrow R_j^K$ if K_i is a foreign key in R_i to R_j, and
- $R_i^Y[Y] \hookrightarrow R_j^K$ for every foreign key Y in R_i to R_j where $Y \subseteq X_i - K_i$.

In other words, the atomic decomposition of a schema consists of a relation scheme for the primary key, a relation scheme for every foreign key, and a relation scheme for every nonkey attribute. Figure 11.2 shows the atomic decomposition of the schema of Figure 11.1.

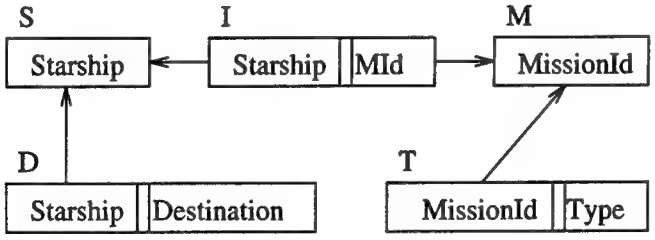


Figure 11.2: An Atomic Decomposition of Schema

From every database b over $\mathcal{B}$ we could construct a unique database b^a over the atomic decomposition $\mathcal{B}^a$ of $\mathcal{B}$ as follows. From every relation $r_i \in b$ over $R_i[X_i, K_i]$ in $\mathcal{B}$, we construct relations $r_i^K = \Pi_{K_i} r_i$, $r_i^Y = \Pi_{K_iY} r_i$, and $r_i^A = \Pi_{K_iA} r_i$ in b^a over R_i^K, R_i^Y, and R_i^A respectively.

Since b and b^a capture the same elementary facts, $\mathcal{B}$ and its atomic decomposition $\mathcal{B}^a$ are semantically equivalent [41]. Notice that every tuple in b is in general broken into several *smaller* tuples in b^a. Therefore every elementary fact captured by b is equivalent to a conjunction of perhaps several smaller elementary facts captured by b^a.

Notice that the atomic decomposition of a database does not contain $\angle$ (by the definition of the total projection operator Π). This implies that null values in a database do not represent elementary facts in a state of the world, which coincides with our intuition.

Furthermore, the atomic decomposition of $\mathcal{B}$ into $\mathcal{B}^a$ is the *finest* possible decomposition, in the sense that every tuple in b^a represents an *atomic* elementary fact whose further decomposition leads to loss of information. For example, tuple (Enterprise, 101) represents the elementary fact "Enterprise is on mission #101", while tuples (Enterprise) and (101) represent the elementary facts "there is a starship named Enterprise" and "there is a starship on mission #101" respectively. The conjunction of the latter two is not equivalent to the former.

11.3 Information Content

Given databases b and b' over schema $\mathcal{B}$ with relations $\{r_i\}_{1\leq i\leq n}$ and $\{r'_i\}_{1\leq i\leq n}$ respectively, $b \cup b'$ denotes the database $\{r_i \cup r'_i\}_{1\leq i\leq n}$ over the same schema. Database b is a *subdatabase* of b', denoted as $b \subseteq b'$, if $r_i \subseteq r'_i$ for $1 \leq i \leq n$. Database b' is *more informative* than b, denoted as $b \sqsubseteq b'$, if the atomic decomposition of b is a subdatabase of the atomic decomposition of b'. In other words, b' is more informative than b if every atomic tuple in b is also an atomic tuple in b'.

Let v, v' be either values in $\mathcal{D}$ or $\angle$. We define the operators $\otimes, \oslash$ on v, v' as follows, where $v \otimes v'$ computes the nonconflicting information in v, v', and $v \oslash v'$ computes the information in v and the nonconflicting information in v':

$$v \otimes v' = \begin{cases} v & \text{if } v = v' \text{ or } v' = \angle \\ v' & \text{if } v = \angle \\ \angle & \text{otherwise} \end{cases}$$

$$v \oslash v' = \begin{cases} v & \text{if } v \neq \angle \\ v' & \text{otherwise} \end{cases}$$

Let t, t' be tuples over X. We define the operators $\otimes, \oslash$ on t, t' as follows, where $t \otimes t'$ computes the nonconflicting information in t, t', and $t \oslash t'$ computes the information in t and the nonconflicting information in t':

$$t \otimes t' = \begin{cases} t & \text{if } t = t' \text{ or } t' = \angle \\ t' & \text{if } t = \angle \\ \angle & \text{otherwise} \end{cases}$$

$$t \oslash t' = \begin{cases} t & \text{if } t \neq \angle \\ t' & \text{otherwise} \end{cases}$$

Let $\mathcal{B} = (\mathcal{R}, \mathcal{C})$ be a schema where $\mathcal{R} = \{R_i[X_i, K_i]\}_{1\leq i\leq n}$. Given relation r_i over $R_i[X_i, K_i]$ and tuples $t, t' \in r_i$, we define the operators $\oplus, \ominus$ on t, t' as follows, where $t \oplus t'$ computes a tuple over X_i that contains the nonconflicting atomic tuples in t, t', and $t \ominus t'$ computes a tuple over X_i that contains the atomic tuples in t and the nonconflicting atomic tuples in t'. Suppose that Z is either K_i, or a foreign key Y in R_i, or a nonkey attribute $A \in X_i$:

$$\begin{aligned} (t \oplus t')[Z] &= t[Z] \otimes t'[Z] \\ (t \ominus t')[Z] &= t[Z] \oslash t'[Z] \end{aligned}$$

Given two relations r_i, r_i' over $R_i[X_i, K_i]$, let $r_i \oplus r_i'$ denote the following relation over $R_i[X_i, K_i]$, which computes the nonconflicting information in r_i, r_i':

$$\{t \oplus t' | t \in r_i \wedge t' \in r_i' \wedge t[K_i] = t'[K_i]\}$$
$$\cup\{t | t \in r_i \wedge \neg(\exists t')(t' \in r_i' \wedge t'[K_i] = t[K_i])\}$$
$$\cup\{t' | t' \in r_i' \wedge \neg(\exists t)(t \in r_i \wedge t[K_i] = t'[K_i])\}$$

and let $r_i \ominus r_i'$ denote the following relation over $R_i[X_i, K_i]$, which computes the information in r_i and the nonconflicting information in r_i':

$$\{t \ominus t' | t \in r_i \wedge t' \in r_i' \wedge t[K_i] = t'[K_i]\}$$
$$\cup\{t | t \in r_i \wedge \neg(\exists t')(t' \in r_i' \wedge t'[K_i] = t[K_i])\}$$
$$\cup\{t' | t' \in r_i' \wedge \neg(\exists t)(t \in r_i \wedge t[K_i] = t'[K_i])\}$$

Given two databases $b = \{r_i\}_{1 \leq i \leq n}$ and $b' = \{r_i'\}_{1 \leq i \leq n}$ over $\mathcal{B}$, let $b \oplus b'$ and $b \ominus b'$ denote respectively the databases $\{r_i \oplus r_i'\}_{1 \leq i \leq n}$ and $\{r_i \ominus r_i'\}_{1 \leq i \leq n}$ over $\mathcal{B}$. Figure 11.3 shows two relations SMD_1 and SMD_2 over the relation scheme SMD of Figure 11.1, together with $SMD_1 \oplus SMD_2$ and $SMD_1 \ominus SMD_2$.

SMD_1

Starship	MId	Destination
Enterprise	101	Rigel
Voyager	102	Talos

SMD_2

Starship	MId	Destination
Enterprise	102	Rigel
Discovery	103	Rigel

$SMD_1 \oplus SMD_2$

Starship	MId	Destination
Enterprise	$\not\perp$	Rigel
Voyager	102	Talos
Discovery	103	Rigel

$SMD_1 \ominus SMD_2$

Starship	MId	Destination
Enterprise	101	Rigel
Voyager	102	Talos
Discovery	103	Rigel

Figure 11.3: Filtering Functions

Theorem 11.1 *For r-valid databases b and b' over $\mathcal{B}$, $b \ominus b'$ is an r-valid database over $\mathcal{B}$ such that $b \sqsubseteq b \ominus b' \sqsubseteq b \cup b'$, and $c \sqsubseteq b \ominus b'$ for every r-valid database c over $\mathcal{B}$ where $b \sqsubseteq c \sqsubseteq b \cup b'$.*

Theorem 11.1 tells us that $b \ominus b'$ is an r-valid database more informative than b but less informative then $b \cup b'$, and is the (unique) most informative such database.[2] In Figure 11.3, $SMD_1 \ominus SMD_2$ is more informative than SMD_1 because it contains the tuple (Discovery, 103, Rigel). It is less informative than $SMD_1 \cup SMD_2$ because it does not contain the atomic tuple (Enterprise, 102).

[2]By restricting ourselves to key-based functional and referential dependencies, such a database always exists, which is not necessarily the case for more general view constraints.

Theorem 11.2 *For r-valid databases b and b' over $\mathcal{B}$, $b \oplus b'$ is an r-valid database over $\mathcal{B}$ such that $b \oplus b' \sqsubseteq b \cup b'$. For every r-valid database c over $\mathcal{B}$ where $b \oplus b' \sqsubset c \sqsubseteq b \cup b'$, there is an r-valid database c' over $\mathcal{B}$ where $b \oplus b' \sqsubset c' \sqsubseteq b \cup b'$, such that neither $c \sqsubseteq c'$ nor $c' \sqsubseteq c$.*

Theorem 11.2 tells us that $b \oplus b'$ is an r-valid database less informative than $b \cup b'$. Moreover, any such database that is strictly more informative than $b \oplus b'$ has to involve a random choice: there is another such database that is incomparable in information content. In other words, $b \oplus b'$ is the (unique) most informative such database that does not involve random choices.[2] In Figure 11.3, $\text{SMD}_1 \oplus \text{SMD}_2$ is less informative than $\text{SMD}_1 \cup \text{SMD}_2$ because the Mission of Enterprise is missing. Any r-valid database strictly more informative than $\text{SMD}_1 \oplus \text{SMD}_2$ but no more informative than $\text{SMD}_1 \cup \text{SMD}_2$ has to contain either (Enterprise, 101) or (Enterprise, 102) but not both, which involves a random choice between the two.

Chapter 12

Interpretation Policy

An *interpretation policy* maps a multilevel database to a multilevel theory and a multilevel structure of the theory. Through this mapping, the superficial syntactic difference in object labels is abstracted away, and the semantic difference hidden in object labels is made precise. As a consequence, the interpretation policy makes it possible to compare the semantics of multiple MAC policies.

Here, we investigate the interpretation policies for two most common multilevel databases, namely multilevel databases with tuple-level and element-level labeling, where objects are tuples and elements in tuples respectively.

Intuitively, element-level labeling seems to be more expressive than tuple-level labeling, because it is more fine-grained in capturing classified elementary facts in a multilevel state of the world. On the other hand, element-level labeling seems to be more complicated and difficult to implement than tuple-level labeling. A formal characterization of the expressive power of these labeling mechanisms would be invaluable in making design decisions such as which mechanism to use in building a multilevel database.

12.1 Tuple-Level Labeling

We first define the multilevel relational model. A *multilevel relation scheme* is a pair $(R[X,K],\mathcal{L})$, where $R[X,K]$ is a relation scheme and $\mathcal{L}$ is a security lattice. A *multilevel database schema* is a pair $(\mathcal{B},\mathcal{L})$, where $\mathcal{B}$ is a database schema and $\mathcal{L}$ is a security lattice.

Let $(\mathcal{B},\mathcal{L})$ be a multilevel schema, where $\mathcal{B}=(\mathcal{R},\mathcal{C})$, $\mathcal{R}=\{R_i[X_i,K_i]\}_{1\leq i\leq n}$, and $\mathcal{L}=(L,\preceq)$. A *multilevel relation with tuple-level labeling* over multilevel relation scheme $(R_i[X_i,K_i],\mathcal{L})$ is a pair (r_i,κ_i), where r_i is a relation over $R_i[X_i,K_i]$ and κ_i is a mapping from tuples over X_i to sets of levels in L, such that $\kappa_i(t)=\{\}$ if and only if $t\notin r_i$, and $l\in\kappa_i(t)$ if t is labeled at $l\in L$.

A *multilevel database with tuple-level labeling* over multilevel database schema $(\mathcal{B},\mathcal{L})$ is a family $\{(r_i,\kappa_i)\}_{1\leq i\leq n}$, where (r_i,κ_i) is a multilevel relation with tuple-level labeling over $(R_i[X_i,K_i],\mathcal{L})$. We denote it by the pair (b,κ), where $b=\{r_i\}_{1\leq i\leq n}$ is a database over $\mathcal{B}$, and $\kappa=\{\kappa_i\}_{1\leq i\leq n}$ is a family of mappings. Figure 12.1 shows a multilevel database over the schema of Figure 11.1 and the security lattice of Figure 10.1. The labels of every tuple are listed to the right of that tuple.

Starship	Mission	Destination	
Enterprise	101	$\mathcal{L}$	$\top$
Enterprise	102	Rigel	m_1
Enterprise	103	Rigel	m_2
Voyager	102	Rigel	m_1
Voyager	102	Talos	m_2
Discovery	103	Rigel	$\bot$

MissionId	Type	
101	spy	m_1
101	mine	m_2
102	explore	m_1, m_2
103	mine	$\bot$

Figure 12.1: A Multilevel Database with Tuple-Level Labeling

The interpretation policy of tuple-level labeling is straightforward. Because every tuple in a relation represents an elementary fact in a state of the world, the label of the tuple naturally represents the classification of the elementary fact.

Since functional dependencies are visible at all levels, a multilevel relation (r_i, κ_i) should satisfy the *polyinstantiation security property*:

4 for every pair of tuples $t, t' \in r_i$ and level l where $l \in \kappa_i(t)$ and $l \in \kappa_i(t')$, $t[K_i] = t'[K_i]$ implies $t = t'$.

In other words, tuples labeled at the same level satisfy all the functional dependencies. A multilevel database (b, κ) satisfies the polyinstantiation security property if every multilevel relation in (b, κ) does. Through the interpretation policy, polyinstantiation security properties are easily mapped to labeling constraints. For example, the polyinstantiation security property over relation scheme MT of Figure 11.1 and the lattice of Figure 10.1 is mapped to:

$$(\forall l \in \mathcal{L})(\forall x, y, z)(\mathrm{MT}^l(x, y) \wedge \mathrm{MT}^l(x, z) \rightarrow y = z).$$

Since a referential dependency $R_i[Y] \hookrightarrow R_j$ represents a relationship between tuples in multilevel relations, and knowing a relationship between two tuples requires knowing the two tuples first, multilevel relations (r_i, κ_i) and (r_j, κ_j) should satisfy the *referential security property*:

5 for every tuple $t \in r_i$ and level $l \in \kappa_i(t)$, there is a tuple $t' \in r_j$ and a level $l' \in \kappa_j(t')$ such that $t[Y] = t'[K_j]$ and $l' \preceq^* l$.

In other words, the label of every foreign key tuple dominate the label of the primary key tuple it refers to. A multilevel database (b, κ) satisfies the referential security property if every pair of multilevel relations involved in a referential dependency does. The multilevel database of Figure 12.1 satisfies polyinstantiation and referential security properties. Through the interpretation policy, referential security properties are also easily mapped to labeling constraints. For example, the

referential security property over the schema of Figure 11.1 and the lattice of Figure 10.1 is mapped to:

$$(\forall l_1 \in \mathcal{L})(\forall x, y, z)(\mathrm{SMD}^{l_1}(x, y, z) \rightarrow (\exists l_2 \in \mathcal{L})(\exists w)(l_2 \preceq l_1 \wedge \mathrm{MT}^{l_2}(y, w))).$$

12.2 Element-Level Labeling

Let $(\mathcal{B}, \mathcal{L})$ be a multilevel schema, where $\mathcal{B} = (\mathcal{R}, \mathcal{C})$, $\mathcal{R} = \{R_i[X_i, K_i]\}_{1 \leq i \leq n}$, and $\mathcal{L} = (L, \preceq)$. A *multilevel relation with element-level labeling* over multilevel relation scheme $(R_i[X_i, K_i], \mathcal{L})$ is a pair (r_i, θ_i), where r_i is a relation over $R_i[X_i, K_i]$ and θ_i is a mapping from attributes in X_i and tuples over X_i to sets of levels in L, such that $\theta_i(A, t) = \{\}$ if and only if $t \notin r_i$ or $t[A] = \mathcal{L}$[1], and $l \in \theta_i(A, t)$ if $t[A]$ is labeled at $l \in L$. When $\theta_i(A, t) = \theta_i(A', t)$ for all $A, A' \in X_i$, we denote $\theta_i(A, t)$ by $\theta_i(X_i, t)$.

A *multilevel database with element-level labeling* over multilevel database schema $(\mathcal{B}, \mathcal{L})$ is a family $\{(r_i, \theta_i)\}_{1 \leq i \leq n}$, where (r_i, θ_i) is a multilevel relation with element-level labeling over $(R_i[X_i, K_i], \mathcal{L})$. We denote it by the pair (b, θ), where $b = \{r_i\}_{1 \leq i \leq n}$ is a database over $\mathcal{B}$, and $\theta = \{\theta_i\}_{1 \leq i \leq n}$ is a family of mappings. Figure 12.2 shows a multilevel database over the schema of Figure 11.1 and the security lattice of Figure 10.1. The labels of every element are listed as a superscript of that element.

Starship	Mission	Destination
Enterprise$^{\perp}$	101$^{\top}$	Rigel$^{\perp}$
Voyager$^{\perp}$	102$^{\perp}$	Rigel^{m_1}
Voyager$^{\perp}$	$\mathcal{L}$	Talos^{m_2}
Discovery$^{\perp}$	103$^{\perp}$	Rigel$^{\perp}$

MissionId	Type
101^{m_1}	spy^{m_1}
102$^{\perp}$	explore$^{m_1, m_2}$
103$^{\perp}$	mine$^{\perp}$

Figure 12.2: A Multilevel Database with Element-Level Labeling

As we observed in Chapter 9, the interpretation policy of element-level labeling is problematic. Since elements in tuples of a database do not have direct correspondence to elementary facts in a state of the world, it is unclear what the correspondence is between the label of an element in a tuple and the classification of any elementary fact. To formulate a natural interpretation policy and the necessary security properties of element-level labeling, we utilize the concept of atomic decomposition from Section 11.2.

Let (b, θ) be a multilevel database with element-level labeling over $(\mathcal{B}, \mathcal{L})$. Consider the atomic decomposition $\mathcal{B}^a$ of $\mathcal{B}$ and the atomic decomposition b^a of b. Notice that both b and b^a capture exactly the same elementary facts, and every elementary fact captured in b is a conjunction of

[1]Null values are not labeled, which is natural because they do not represent elementary facts in a state of the world.

several elementary facts captured in b^a. Hence we define the interpretation policy of (b, θ) by the interpretation policy of (b^a, κ) — a multilevel database with tuple-level labeling over $(\mathcal{B}^a, \mathcal{L})$. In particular, for every relation $r_i \in b$, tuple $t \in r_i$, and attribute $A \in X_i$:

$$\theta_i(A,t) = \begin{cases} \kappa_i^K(t[K_i]) & \text{if } A \in K_i \\ \kappa_i^Y(t[K_iY]) & \text{if } A \in Y \text{ and } t[Y] \neq \angle \\ \kappa_i^A(t[K_iA]) & \text{if } t[A] \neq \angle \end{cases}$$

Informally the interpretation policy says that the labels on primary keys classify their existence, and the labels on foreign keys and nonkey attribute values classify their relationships with primary keys. From this definition, we derive the *key classification property* of element-level labeling:

6.1 for every tuple $t \in r_i$ and attributes $A, A' \in K_i$, $\theta_i(A,t) = \theta_i(A',t)$, and

6.2 for every tuple $t \in r_i$, foreign key Y in R_i, and attributes $A, A' \in Y$, $\theta_i(A,t) = \theta_i(A',t)$.

In other words, primary and foreign keys are labeled uniformly.

From the polyinstantiation security property of tuple-level labeling, we know that $t[K_i] = t'[K_i]$ and $\kappa_i^Y(t) = \kappa_i^Y(t')$ implies $t = t'$ for all $t, t' \in r_i^Y$. Similarly $t[K_i] = t'[K_i]$ and $\kappa_i^A(t) = \kappa_i^A(t')$ implies $t = t'$ for all $t, t' \in r_i^A$. Hence we derive the *polyinstantiation security property* of element-level labeling:

7 for every pair of tuples $t, t' \in r_i$ and attribute $A \in X_i - K_i$, we have that $t[K_i] = t'[K_i], \theta_i(K_i,t) = \theta_i(K_i,t')$, and $\theta_i(A,t) = \theta_i(A,t')$ implies $t[A] = t'[A]$.

In other words, foreign keys or nonkey attribute values, which are labeled at the same level and correspond to the same primary keys, are identical.

From the referential security property of tuple-level labeling and referential dependencies $R_i^Y[K_i] \hookrightarrow R_i^K, R_i^A[K_i] \hookrightarrow R_i^K$ in $\mathcal{C}^a$, we know that $t[K_i] = t'$ implies $\kappa_i^K(t') \preceq \kappa_i^Y(t)$ for all $t \in r_i^Y, t' \in r_i^K$; and $t[K_i] = t'$ implies $\kappa_i^K(t') \preceq \kappa_i^A(t)$ for all $t \in r_i^A, t' \in r_i^K$. Hence we derive the *primary key security property* of element-level labeling:

8 for every tuple $t \in r_i$ and $A \in X_i - K_i$ where $t[A] \neq \angle$, $\theta_i(K_i,t) \preceq \theta_i(A,t)$.

In other words, primary keys are dominated by foreign keys and nonkey attribute values.

Again from the referential security property of tuple-level labeling and the referential dependency $R_i^Y[Y] \hookrightarrow R_j^K$ in $\mathcal{C}^a$, we know that $t[Y] = t'$ implies $\kappa_j^K(t') \preceq \kappa_i^Y(t)$ for all $t \in r_i^Y, t' \in r_j^K$. Hence we derive the *foreign key security property* of element-level labeling for every referential dependency $R_i[Y] \hookrightarrow R_j$ in $\mathcal{C}$:

9 for every tuple $t \in r_i$ where $t[Y] \neq \angle$, there is $t' \in r_j$ such that $t[Y] = t'[K_j]$ and $\theta_j(K_j,t') \preceq \theta_i(Y,t)$.

In other words, the label of every foreign key value dominates the label of the primary key value it refers to.

All properties defined in this section have been identified in the literature as desirable, indicating that our interpretation policy for multilevel databases with element-level labeling is natural. In

particular, our properties 2.1, 6.1, and 8 together form the entity integrity as defined in [22, 29]. Hence our definition of the interpretation policy of element-level labeling provides a semantic justification of entity integrity. With the natural requirement that null values are not labeled, our property 7 is equivalent to the PI-FD property of [44]. Our properties 3.1, 6.2, and 9 together provide a formal definition and semantic justification of referential integrity in the multilevel relational model with element-level labeling.

Figure 12.3 shows a multilevel database with tuple-level labeling, over the schema of Figure 11.2. It is semantically equivalent to the multilevel database of Figure 12.2, because the two are mapped to the same multilevel theory and structure according to our interpretation policies. The null value in Figure 12.2 has disappeared in Figure 12.3.

Starship	
Enterprise	$\bot$
Voyager	$\bot$
Discovery	$\bot$

Starship	MId	
Enterprise	101	$\top$
Voyager	102	$\bot$
Discovery	103	$\bot$

Starship	Destination	
Enterprise	Rigel	$\bot$
Voyager	Rigel	m_1
Voyager	Talos	m_2
Discovery	Rigel	$\bot$

MissionId	
101	m_1
102	$\bot$
103	$\bot$

MissionId	Type	
101	spy	m_1
102	explore	m_1, m_2
103	mine	$\bot$

Figure 12.3: An Atomic Decomposition of Database

12.3 Design Trade-Off

From our interpretation policies of tuple-level and element-level labeling, we can conclude that tuple-level and element-level labeling mechanisms have exactly the same expressive power, because for every multilevel database with element-level labeling, we can find a multilevel database with tuple-level labeling that captures exactly the same information in a multilevel state of the world, and vice versa.

But trade-off does exist between tuple-level and element-level labeling mechanisms when designing a multilevel database. On one hand, element-level labeling is more complicated than tuple-level labeling, since labels are attached to elements rather than tuples. On the other hand, tuple-level labeling requires more complicated schemas to capture the same amount of information as element-level labeling, making query and update more expensive because the same elementary fact captured by one tuple with element-level labeling is captured by several tuples with tuple-level labeling.

To simplify discussions, we restrict ourselves to multilevel databases with tuple-level labeling in the rest of the report. Because of the equivalence of expressive power between tuple-level and

element-level labeling mechanisms, the results can be easily generalized to multilevel databases with element-level labeling.

Chapter 13

View Policy

A *view policy* consists of a set of view constraints and a specification of the validity of view constraints. This policy specifies the upward information flow requirements for the view constraints, which indicates the believability of low data at high levels.

13.1 Sample View Policies

According to the Bell-LaPadula model, low data are always visible at high. However, since low data could contradict high data, *visibility* should be distinguished from *believability*.

The filter function [21, 29] and the security logic [16] proposed in the literature take one extreme position by equating believability to visibility, thus maximizing believability. However, integrity is compromised if a low tuple contradicts some high tuples with respect to the constraints, which leads to an invalid high database. For example, consider the following multilevel relation over the schema of Figure 11.1 and the lattice of Figure 10.1:

Starship	Mission	Destination	
Enterprise	$\angle$	Talos	$\top$
Enterprise	102	Rigel	m_1
Enterprise	103	Rigel	m_2

When querying the mission of Enterprise at level $\top$, users will get back both 102 and 103, which contradicts the constraint "starships have unique missions".

Smith and Winslett proposed a belief-based semantics of the multilevel relational model [53], which defines a multilevel relational database as a set of unrelated single-level relational databases, one for every level. They made a clear distinction between visibility and believability, and took the other extreme position by allowing no low tuples to be believable at high, thus minimizing

believability. Their semantics serves as a nice framework within which other semantics could be compared. However a multilevel relational database that directly employs their semantics would no longer be multilevel — it would be a set of single-level relational databases in which there is no upward information flow across levels. For example, consider the following multilevel relation over the schema of Figure 11.1 and the lattice of Figure 10.1:

Starship	Mission	Destination	
Enterprise	102	Rigel	$\bot$

When querying the mission of Enterprise at level $\top$, users will get back an empty answer, because no information about Enterprise is considered believable at that level.

Thuraisingham first formalized the distinction between visibility and believability by a proof-theoretic semantics of the multilevel relational model [55], which consists of a nonmonotonic inference rule stating that low data are believable at high as long as they do not contradict high data. Given two low tuples labeled incomparably, what happens if either tuple does not contradict high data, but their combination does? To determine what is believable at high, the result of Thuraisingham's approach would depend on the (random) order in which the nonmonotonic inference rule is applied to these two tuples, which introduces ambiguity. For example, consider the following multilevel relation over the schema of Figure 11.1 and the lattice of Figure 10.1:

Starship	Mission	Destination	
Enterprise	102	Rigel	m_1
Enterprise	103	Talos	m_2

When querying the mission of Enterprise at level $\top$, users will get back either 102 or 103 but not both. It should be noticed that such problems occur even with a totally ordered security lattice, if we allow arbitrary constraints. For example, a constraint could state that there should be no more than two starships going to Rigel. If we have one high tuple (Enterprise, 101, Rigel) together with two low tuples (Voyager, 102, Rigel) and (Discovery, 103, Rigel), then at most one low tuple is believable at high, but it is unclear which one should be.

In the rest of this chapter, we develop a view policy for multilevel databases with tuple-level labeling where the view constraints consist of key-based functional and referential dependencies, which overcomes the above-mentioned problems.

13.2 Validity and Views

Intuitively a database represents one view of a state of the world (perceived world [35]), while a multilevel database represents multiple views of a multilevel state of the world — one for every

level. Furthermore, these multiple views are related by the security lattice. Contained in the view at a level are tuples believable at that level. Consequently integrity should be enforced within each view, rather than across multiple views.

What tuples belong to the view at a level? First, all tuples labeled at that level should be part of the view. Second, for tuples labeled at a lower level, as many of them as possible should be part of the view as long as integrity is preserved, in order to maximize sharing. Third, in case that either but not both of two low tuples could be in a high view, neither of them should be in the high view, because the high view lacks further information to justify the preference of one over the other. In other words, view constraints serve as a filter on how much low data could flow high.

Formally, recall from Section 12.1 that a multilevel schema is a pair $(\mathcal{B}, \mathcal{L})$, where $\mathcal{B} = (\mathcal{R}, \mathcal{C})$, $\mathcal{R} = \{R_i[X_i, K_i]\}_{1 \leq i \leq n}$, and $\mathcal{L} = (L, \preceq)$. Let $(b, \kappa) = \{(r_i, \kappa_i)\}_{1 \leq i \leq n}$ be a multilevel database over $(\mathcal{B}, \mathcal{L})$, and $l \in L$ be a level.

The *l-slice* of multilevel relation (r_i, κ_i), denoted as $\sigma_l(r_i, \kappa_i)$, is a subrelation of r_i defined as $\{t | t \in r_i \wedge l \in \kappa_i(t)\}$. The *l-slice* of multilevel database (b, κ), denoted as $\sigma_l(b, \kappa)$, is a subdatabase of b defined as $\{\sigma_l(r_i, \kappa_i)\}_{1 \leq i \leq n}$. The l-slices $\sigma_l(r_i, \kappa_i)$ and $\sigma_l(b, \kappa)$ are respectively a single-level relation and database collecting all tuples in r_i and b that are labeled at l. The $\top$-slice of the multilevel database of Figure 12.1 consists of the first tuple in SMD and no tuples of MT.

The *l-validity* of (b, κ) and the *l-view* of (b, κ), denoted as $\varsigma_l(b, \kappa) = \{\varsigma_l(r_i, \kappa_i)\}_{1 \leq i \leq n}$ where $\varsigma_l(r_i, \kappa_i)$ is the *l-view* of (r_i, κ_i), are defined recursively as follows.

- Suppose that l is the bottom level of $\mathcal{L}$. Define $\varsigma_l(b, \kappa) = \sigma_l(b, \kappa)$. If $\sigma_l(b, \kappa)$ is valid, then (b, κ) is l-valid. Otherwise (b, κ) is not l-valid.

- Suppose that l is not the bottom level of $\mathcal{L}$. Let b_H be $\sigma_l(b, \kappa)$ and b_L be $\oplus_{l' \prec l} \varsigma_{l'}(b, \kappa)$. Also let b_U be $b_H \ominus b_L$. Define $\varsigma_l(b, \kappa) = b_U$. If b_U is valid then (b, κ) is l-valid. Otherwise (b, κ) is not l-valid.

Multilevel database (b, κ) is *valid* if it is l-valid for every level $l \in L$. Notice that the l-views of (r_i, κ_i) and (b, κ) are respectively a single-level relation and database.

The above definition formalizes our intuition about views. All atomic tuples labeled at level l are part of the l-view since $b_H \sqsubseteq \varsigma_l(b, \kappa)$ according to Theorem 11.1. For atomic tuples labeled below l, as many of them as possible are part of the l-view since they are part of b_U according to Theorem 11.1. In case that either but not both of two atomic tuples labeled below l could be in the l-view, neither is in the l-view since neither would be in b_L according to Theorem 11.2. The multilevel database of Figure 12.1 is valid, and its $\top$-view is shown in Figure 13.1.

Our view policy coincides with the Bell-LaPadula model for primary key values, in the sense that all low primary key values are part of the high view. In the extreme case that there are no view constraints (i.e., $X_i = K_i$ for $1 \leq i \leq n$ and $\mathcal{C}$ is empty), our view policy completely coincides with the Bell-LaPadula model, in the sense that all low tuples are part of the high view, since $\varsigma_l(b, \kappa) = \bigcup_{l' \preceq l} \sigma_{l'}(b, \kappa)$.

Our view policy is an extension of the Bell-LaPadula model, in the sense that we distinguish between two kinds of low data: those that are believable at high (e.g., the atomic tuple (Enterprise, Rigel)), and those that are visible but not believable at high (because they violate view constraints

Starship	Mission	Destination
Enterprise	101	Rigel
Voyager	102	$\mathcal{L}$
Discovery	103	Rigel

MissionId	Type
101	$\mathcal{L}$
102	explore
103	mine

Figure 13.1: A $\top$-View

when combined with high data) (e.g., the atomic tuple (Enterprise, 102)). Integrity is enforced only on believable low data.

Let f_l be the number of levels immediately dominated by l, and f be $\max\{f_l | l \in L\}$. For multilevel relation (r_i, κ_i), the size of its l-slice $\sigma_l(r_i, \kappa_i)$ and l-view $\varsigma_l(r_i, \kappa_i)$ is bounded by $O(|r_i|)$. The cost of computing $\varsigma_l(r_i, \kappa_i)$ for bottom level is bounded by $O(|L| \times |r_i|)$. For nonbottom level l, the cost of computing b_H is bounded by $O(|L| \times |r_i|)$, the cost of computing b_L is bounded by $O(|r_i|^f)$, and the cost of computing b_U is bounded by $O(|r_i|^2)$. Assuming that $|L| \ll |r_i|$ and $2 \ll f$, the cost of computing $\varsigma_l(r_i, \kappa_i)$ is bounded by $O(|L_l| \times |r_i|^f)$, where L_l is the set of levels dominated by l.

13.3 Validity Checking

Given a multilevel database, straightforward validity checking based on the recursive definition of validity is likely to be expensive, because it involves computing views for all levels and checking their validity. Luckily, multilevel validity could be equivalently characterized by multilevel security properties, whose computation is comparable in complexity to integrity checking in single-level databases.

Theorem 13.1 *A multilevel database is valid if and only if it satisfies polyinstantiation and referential security properties.*

Theorem 13.1 tells us that view computation is not necessary for validity checking. For example, the multilevel database of Figure 12.1 is valid because it satisfies polyinstantiation and referential security properties. Furthermore, validity checking in multilevel databases is comparable in complexity to that in single-level databases. The cost of checking polyinstantiation security in multilevel relation (r_i, κ_i) is bounded by $O(|L|^2 \times |r_i|^2)$, while the cost of checking key integrity in single-level relation r_i is bounded by $O(|r_i|^2)$. For $R_i[Y] \hookrightarrow R_j$ in $\mathcal{C}$, the cost of checking referential security in multilevel database (b, κ) is bounded by $O(|L|^3 \times |r_i| \times |r_j|)$, while the cost of checking referential integrity in single-level database b is bounded by $O(|r_i| \times |r_j|)$.

13.4 Validity Enforcement

According to the Bell-LaPadula model, low updates could only affect views at comparably higher levels. For the classes of constraints in our multilevel relational model, a low insertion will not invalidate any high views, because the inserted low tuple could either cause other low tuples to be removed from a high view, or be excluded from a high view by some high tuples. The only situation in which a low deletion will invalidate a high view is when the deleted low tuple is referred to by some high tuple through a referential dependency.

Intuitively, if a low deletion invalidates a high view, it should not be aborted. Instead it should be extended with necessary compensating updates in order to enforce both integrity and secrecy. Not all compensating updates are acceptable. They should have at least the following three natural properties:

1. The compensating updates should be at comparably high levels, because of the $*$-property of the Bell-LaPadula model.

2. The compensating high updates should not cause a loss of high data. In other words, a low deletion should only be extended with high insertions.

3. The amount of high data added through compensating high insertions should be minimized.

Let $(b, \kappa) = \{(r_i, \kappa_i)\}_{1 \le i \le n}$ be a valid multilevel database over multilevel schema $(\mathcal{B}, \mathcal{L})$, where $\mathcal{B} = (\mathcal{R}, \mathcal{C})$, $\mathcal{R} = \{R_i[X_i, K_i]\}_{1 \le i \le n}$, and $\mathcal{L} = (L, \preceq)$. Also let $l \in L$ be a level and t be a tuple over X_i. An *update* at level l has the form $\text{insert}_j^l(t)$ or $\text{delete}_j^l(t)$, which specifies respectively the insertion to or deletion from multilevel relation (r_j, κ_j) of tuple t at level l.

Let $\text{op}_j^l(t)$ be an update at level l where op is either insert or delete. Define $(b', \kappa') = \{(r_i', \kappa_i')\}_{1 \le i \le n}$ to be a multilevel database over $(\mathcal{B}, \mathcal{L})$, where $(r_i', \kappa_i') = (r_i, \kappa_i)$ for all $i \ne j$, and $\kappa_j'(t') = \kappa_j(t')$ for all $t' \ne t$. Define r_j' and $\kappa_j'(t)$ as follows.

1. op = insert

   (a) $r_j' = r_j \cup \{t\}$, and

   (b) $\kappa_j'(t) = \kappa_j(t) \cup \{l\}$.

2. op = delete

   (a) $\kappa_j'(t) = \kappa_j(t) - \{l\}$, and

   (b) $r_j' = r_j - \{t\}$ if $\kappa_j(t) = \{l\}$.

The update $\text{op}_j^l(t)$ applied to multilevel database (b, κ) is *correct* if and only if multilevel database (b', κ') is l-valid. If the update is correct, then its result is a multilevel database $(\tilde{b}, \tilde{\kappa}) = \{(\tilde{r}_i, \tilde{\kappa}_i)\}_{1 \le i \le n}$ defined as follows. Let t' be the tuple over X_j where $t'[K_j] = t[K_j]$ and $t'[X_j - K_j] = \mathcal{L}$.

1. If op = insert, then $(\tilde{b}, \tilde{\kappa}) = (b', \kappa')$.

2. If op = delete, then $(\tilde{r}_i, \tilde{\kappa}_i) = (r'_i, \kappa'_i)$ for all $i \neq j$, and $\tilde{\kappa}_j(\tilde{t}) = \kappa'_j(\tilde{t})$ for all $\tilde{t} \neq t'$. Let L' be the set of levels $l' \in L$ such that $l \prec^* l'$, (b', κ') is not l'-valid, and (b', κ') is $\tilde{l}$-valid for every $\tilde{l} \in L$ where $l \preceq^* \tilde{l} \prec^* l'$.

   (a) If $L' = \{\}$, then $(\tilde{b}, \tilde{\kappa}) = (b', \kappa')$.

   (b) If $L' \neq \{\}$, then $\tilde{r}_j = r'_j \cup \{t'\}$ and $\tilde{\kappa}_j(t') = \kappa'_j(t') \cup L'$.

For example, deleting the first two MT tuples consecutively (in either order) from the multilevel database of Figure 12.1 would lead to inserting the MT tuple (101, $\angle$) at $\top$.

Theorem 13.2 *If the update is correct, then the result of the update is valid.*

Theorem 13.2 tells us that the extended update preserves the validity of multilevel databases. It is also secure because correct low deletions will not be aborted.

The extended update also satisfies the three properties identified earlier, for the following reasons.

1. The compensating updates are at comparably high levels. For any level l' where $l \not\preceq^* l'$, the l'-view after an update at l remains the same. The l-view after an update at l differs from that before the update precisely in the effect of the update.

2. The compensating high updates do not cause a loss of high data. For any level l' where $l \prec^* l'$, the necessary compensating high insertions at l' are performed to restore referential security, if it is violated by a deletion at level l.

3. The amount of high data added through compensating high insertions is minimized. First, for any level l' where $l \prec^* l'$, compensating insertions are performed at l' only if a deletion at level l invalidates the l'-view. Second, the number of levels at which compensating insertions are performed is minimized by the definition of L'. Third, the information content of the tuple to be inserted by compensating insertions is minimized by the definition of t'.

It is easy to see that view computation is not necessary for validity enforcement. When deleting tuple t labeled at level l in relation r_i, the set of tuples $\tilde{t}$ that refer to t through referential dependencies could be computed as a by-product of checking the referential security property. The set of minimal levels l' of $\tilde{t}$ such that $l \prec l'$, namely L', is the set of levels at which the primary key value of t, namely t', needs to be inserted.

Chapter 14

Update Policy

An *update policy* consists of a set of labeling constraints, a set of updates, and a specification of the enforcement of labeling constraints in performing the updates. This policy specifies the mechanisms to eliminate inference channels in the enforcement of labeling constraints.

14.1 Sample Update Policies

We consider the restricted-value policy of [43] and the insert-low policy of [62], both of which are designed to eliminate inference channels in the enforcement of the no-polyinstantiation constraint. For easy presentation, we adapt these policies to the context of multilevel databases with tuple-level labeling. The no-polyinstantiation constraint states:

Two distinct tuples cannot have identical primary key values.

If low users insert a tuple which has the same primary key value as an existing high tuple, then either the low insertion has to be rejected, leading low users to infer the existence of the high tuple, or the high tuple has to be overwritten, causing a loss of high data. Similarly, if high users insert a tuple which has the same primary key value as an existing low tuple, then either the low tuple has to be deleted, leading low users to infer the existence of the high tuple, or the high insertion has to be rejected, causing a loss of high data.

The example below illustrates how the restricted-value policy removes this dynamic inference channel in the no-polyinstantiation constraint. Consider the following multilevel relation over the schema of Figure 11.1 and the lattice of Figure 10.1:

Starship	Mission	Destination	
Enterprise	102	Rigel	$\bot$

When users try to replace 102 by 101 at level $\top$, the update is extended to:

1. Replace 102 by $\surd$ at level $\bot$.

2. Insert (Enterprise, 101, Rigel) at level $\top$.

The extended update ensures no-polyinstantiation at the price of introducing a (partial) static inference channel, because users at level $\bot$ can infer from the restricted-value $\surd$ that Enterprise has a high mission. Moreover, the high update is extended with a low insertion, which is against the spirit of the $*$-property of the Bell-LaPadula model.

The example below illustrates how the insert-low policy removes this dynamic inference channel in the no-polyinstantiation constraint. Consider the following multilevel relation over the schema of Figure 11.1 and the lattice of Figure 10.1:

Starship	Mission	Destination	
Enterprise	101	Rigel	$\top$

When users try to insert tuple (Enterprise, 102, Rigel) at level $\bot$, the update is extended to:

1. Delete (Enterprise, 101, Rigel) at level $\top$.

2. Insert (Enterprise, 102, Rigel) at level $\bot$.

The extended update ensures no-polyinstantiation at the price of losing high data.

In the rest of this chapter, we characterize the desirable properties of update policies for multilevel databases with tuple-level labeling, which do not suffer from the above-mentioned problems.

14.2 Polarity and Force

For easy presentation of our results, we borrow two standard syntactic notions in first-order logic from [31]. For every formula α, we assign a *polarity* to every subformula in α, which is either *positive* or *negative*. The polarity of a subformula in α provides a syntactic indication as to how the truth of the subformula relates to the truth of α. The polarity of any subformula in α is determined by the following rules:

1. α has positive polarity.

2. If α has the form $\neg\beta$, then β has polarity opposite to α.

3. If α has the form $\beta \wedge \gamma$ or $\beta \vee \gamma$, then β and γ have the same polarity as α.

4. If α has the form $\beta \rightarrow \gamma$, then β has polarity opposite to α and γ has the same polarity as α.

5. If α has the form $(\forall x)\beta$ or $(\exists x)\beta$, then β has the same polarity as α.

Based on the polarities of subformulas in α, we assign a *force* to every quantifier in α, which is either *universal* or *existential*. The force of a quantifier in α gives a syntactic indication of what role the quantifier has towards the truth of α. For a subformula β in α of the form $(\forall x)\gamma$ or $(\exists x)\gamma$, the force of the outmost quantifier in β is determined by the following rules:

1. The quantifier has universal force if
   (a) it is a universal quantifier and β has positive polarity; or
   (b) it is an existential quantifier and β has negative polarity.
2. The quantifier has existential force if
   (a) it is an existential quantifier and β has positive polarity; or
   (b) it is a universal quantifier and β has negative polarity.

14.3 Labeling Constraints

Now we are ready to define labeling constraints in the multilevel relational model. *Level expressions* are defined recursively as follows:

- Every level l in $\mathcal{L}$ is a level expression.
- Given level expressions l_1 and l_2, $l_1 \sqcup l_2$ and $l_1 \sqcap l_2$ are level expressions denoting respectively the least upper bound and the greatest lower bound of l_1 and l_2.

Given level expressions l_1 and l_2 where $l_1 \preceq l_2$, a *lattice expression* has the form $\mathcal{L}_{l_1}^{l_2}$, denoting the sublattice of $\mathcal{L}$ whose bottom and top levels are l_1 and l_2 respectively.

Recall from Section 12.1 that a multilevel schema is a pair $(\mathcal{B}, \mathcal{L})$, where $\mathcal{B} = (\mathcal{R}, \mathcal{C})$, $\mathcal{R} = \{R_i[X_i, K_i]\}_{1 \le i \le n}$, and $\mathcal{L} = (L, \preceq)$. A *labeling constraint* is a sentence in many-sorted first-order predicate calculus. There is a domain sort and a level sort. Predicate symbols include domain equality and R_i^x where x is a level variable. Non-equality atomic formulas are *relational formulas*. Quantifiers of the level sort have the form $(Qx \in L)$, where Q is either $\forall$ or $\exists$, and L is a lattice expression whose bottom level is either $\bot$ or a variable with $\bot$ as the default, and whose top level is either $\top$ or a variable with $\top$ as the default.[1] A labeling constraint over the schema of Figure 11.1 and the lattice of Figure 10.1 might be:

$$(\forall l \in \mathcal{L})(\forall x, y, z)(\mathrm{MT}^l(x, y) \wedge \mathrm{MT}^l(x, z) \rightarrow y = z) \tag{14.1}$$

which states the polyinstantiation security property of MT, namely MT tuples labeled at the same level have unique MissionId values. Another labeling constraint over the same schema and lattice might be:

$$(\forall l_1 \in \mathcal{L})(\forall x, y, z)(\mathrm{SMD}^{l_1}(x, y, z) \rightarrow (\exists l_2 \in \mathcal{L}^{l_1})(\exists w)\mathrm{MT}^{l_2}(y, w)) \tag{14.2}$$

[1] We do not allow arbitrary level constants in labeling constraints. This makes the specification of constraints independent of specific lattices. It also simplifies the definition of l-validity for labeling constraints later. Nevertheless, the results presented here could easily be generalized to labeling constraints containing arbitrary level constants.

which states the referential security property from SMD to MT, namely every SMD tuple refers to an MT tuple at the same or a lower level [37].

A labeling constraint α is *single-level* if it has the form $(\forall x \in \mathcal{L})\beta$, where β does not contain level quantifiers. For example, constraint (14.1) is single-level, while constraint (14.2) is not.

Given a level l in $\mathcal{L}$, the *l-instance* of a labeling constraint α is $\alpha[l/\top]$, which is denoted by α^l. Notice that the $\top$-instance of α is α, and the l-instance of α is α if α does not contain the level constant $\top$.

Recall from Section 12.1 that a multilevel database over multilevel schema $(\mathcal{B}, \mathcal{L})$ is a pair (b, κ) where $b = \{r_i\}_{1 \le i \le n}$ and $\kappa = \{\kappa_i\}_{1 \le i \le n}$. Given a level l in $\mathcal{L}$, a labeling constraint α is *l-valid* in (b, κ) if the l-instance of α is true in the first-order structure that assigns the l'-slice $\sigma_{l'}(r_i, \kappa_i)$ to $R_i^{l'}$ for $1 \le i \le n$ and l' in $\mathcal{L}$, which is denoted as $(b, \kappa) \models \alpha^l$. A labeling constraint α is *valid* in (b, κ) if α is $\top$-valid in (b, κ). For example, constraints (14.1) and (14.2) are both m_1-valid and valid in the multilevel database of Figure 12.1.

Since $(b, \kappa) \models \alpha^l$ iff $(b, \kappa)^l \models \alpha^l$ for any level l in $\mathcal{L}$, the l-validity of α could be checked at l, but not at any level lower than or incomparable to l, by accessing only data that are visible at l, namely $(b, \kappa)^l$. Therefore, if α contains the level constant $\top$, then its validity could only be checked at $\top$.

14.4 Static Inference Channels

Static inference channels are found in a particular state of the database, and depend on the data and labeling constraints true in that state of the database. In other words, from low data combined with labeling constraints, the low user could often infer some high information. For example, consider a labeling constraint which requires that every foreign key value refers to a primary key value. If a foreign key value is labeled low but the primary key value it refers to is labeled high, then the low user would see the foreign key value but not the primary key value. Hence, the existence of the primary key value could be inferred by the low user from the foreign key value if he knows about the labeling constraint. Static inference channels through functional and multivalued dependencies were studied in [32, 54].

We identify common classes of labeling constraints whose enforcement is free of static inference channels. Intuitively, a static inference channel exists in a multilevel database if high information could be derived from low tuples together with (the low instances of) labeling constraints. In other words, low tuples combined with labeling constraints logically imply some sentence that is not valid in the low database. Since the multilevel database is known to be valid with respect to labeling constraints, the low user could infer that the implied sentence must be valid in the high database.

Theorem 14.1 *If all level quantifiers in α have universal force, then there is no static inference channel in α.*

As a consequence of Theorem 14.1, there is no static inference channel in single-level labeling constraints. For example, there is no static inference channel in constraint (14.1). In other words, there could not be a valid MT relation containing two tuples at the same level and with the same MissionId value.

Corollary 14.2 *If every level quantifier in α with existential force has the form $(Qy \in L)\beta$, where L is a lattice expression whose top level is not $\top$, then there is no static inference channel in α.*

For example, there is no static inference channel in constraint (14.2) according to Corollary 14.2. In other words, there could not be a valid database that contains an SMD tuple not referring to any lower MT tuple. However, if we relax the requirement by replacing constraint (14.2) with the following labeling constraint (14.3) instead, which states that every SMD tuple refers to an MT tuple:

$$(\forall l_1 \in \mathcal{L})(\forall x, y, z)(\mathrm{SMD}^{l_1}(x, y, z) \rightarrow (\exists l_2 \in \mathcal{L})(\exists w)\mathrm{MT}^{l_2}(y, w)) \tag{14.3}$$

then there are static inference channels at l for every $l \neq \top$. If a low SMD tuple does not refer to any low MT tuple, then it must refer to a high MT tuple.

A natural way of eliminating static inference channels in labeling constraint α is by requiring that α is l-valid for every level l in $\mathcal{L}$. In other words, if labeling constraints are enforced at every level, then there is no static inference channel. Hence we could insist that constraint (14.3) be enforced at every level, namely every SMD tuple has to refer to a visible MT tuple, which is equivalent to enforcing constraint (14.2).

14.5 Dynamic Inference Channels

Dynamic inference channels are found in a particular state transition of the database, and depend on the data and labeling constraints true in the state before the transition as well as the behavior of the database in response to the transition.[2] In other words, the result of a low update could violate some labeling constraints when combined with high data, but prohibiting the low update would enable the low user to infer the existence of relevant high data. For example, consider a labeling constraint which requires that every foreign key value refers to a visible primary key value. If a foreign key value is labeled high but the primary key value it refers to is labeled low, then prohibiting a low deletion of the primary key value would enable the low user to infer the existence of the high foreign key value. Dynamic inference channels through the enforcement of polyinstantiation and referential integrity were studied in [7, 43].

We identify common classes of labeling constraints whose enforcement is free of dynamic inference channels. Intuitively, a dynamic inference channel exists in a multilevel database if the result of a low update is not valid with respect to labeling constraints, even if the low database is valid with respect to (the low instances of) labeling constraints [7, 43]. In other words, a low update results in a valid low database but not a valid high database. Since the multilevel database has to be valid, the low update has to be prohibited, thus enabling the low user to infer the existence of relevant high data.

Theorem 14.3 *If all level quantifiers in α have existential force, then there is no dynamic inference channel in α.*

[2] Because dynamic inference channels involve system behavior, in many cases the mechanisms could also be used as covert signaling channels between cooperating malicious high and low subjects. However, we concern ourselves here only with undesired inferences through such mechanisms, which do not require either a high "sender" or that the low "receiver" be malicious.

For example, there is no dynamic inference channel in the following labeling constraint (14.4) according to Theorem 14.3, which states that there is a level at which no starships are going to Rigel:

$$(\exists l \in \mathcal{L})(\forall x, y, z)(\mathrm{SMD}^l(x, y, z) \rightarrow z \neq \text{Rigel}). \tag{14.4}$$

If there is a level in the low database after a low update at which no starships are going to Rigel, then there is definitely such a level in the entire database after the low update.

Corollary 14.4 *If every level quantifier in α with universal force has the form $(Qy \in L)\beta$, where L is a lattice expression whose top level is not $\top$, then there is no dynamic inference channel in α.*

For example, there is no dynamic inference channel in the following labeling constraint (14.5) according to Corollary 14.4, which states that there is a level at which polyinstantiation is prohibited for the MissionId attribute of MT:

$$(\exists l \in \mathcal{L})(\forall l_1, l_2 \in \mathcal{L}^l)(\forall x, y_1, y_2)(\mathrm{MT}^{l_1}(x, y_1) \wedge \mathrm{MT}^{l_2}(x, y_2) \rightarrow y_1 = y_2). \tag{14.5}$$

If there is a level in the low database after a low update at which polyinstantiation is prohibited for the MissionId attribute of MT, then there is definitely such a level in the entire database after the low update.

Theorem 14.5 *If α is single-level, then there is no dynamic inference channel in α.*

For example, there is no dynamic inference channel in constraint (14.1) according to Theorem 14.5. If no two low MT tuples have the same MissionId value after a low update, and no two high MT tuples have the same MissionId value before the low update, then no two MT tuples at the same level have the same MissionId value after the low update.

It is worth noticing that there is a syntactic symmetry between the classes of labeling constraints whose enforcement is free of static inference channels, such as the $(\forall)^*$-class (Theorem 14.1) and the $(\forall)^*(\exists)^*$-class (Corollary 14.2), and the classes of labeling constraints whose enforcement is free of dynamic inference channels, such as the $(\exists)^*$-class (Theorem 14.3) and the $(\exists)^*(\forall)^*$-class (Corollary 14.4). Moreover, the enforcement of the class of single-level labeling constraints is free of both static and dynamic inference channels.

14.6 Eliminate Inference Channels

We identify common classes of labeling constraints whose enforcement, although not free of dynamic inference channels, can be made free of static and dynamic inference channels if we extend the enforcement by a proper update policy.

When a labeling constraint cannot be enforced because of a dynamic inference channel, the low update could sometimes be extended to a multilevel update that enforces the labeling constraint. Because of the $*$-property of the Bell-LaPadula model, a natural requirement is that the low update should only be extended with updates at high levels. A *removable* dynamic inference channel is one for which such an update policy exists.

Theorem 14.6 *Suppose that labeling constraint α has the form*

$$(\forall x_1 \in L_1, \ldots, x_k \in L_k)(\forall \bar{y})(P_1 \wedge \cdots \wedge P_m \rightarrow (\exists x'_1 \in L'_1, \ldots, x'_{k'} \in L'_{k'})(\exists \bar{y'})(P'_1 \wedge \cdots \wedge P'_{m'}))$$

where $P_1, \ldots, P_m$ are relational formulas, $P'_1, \ldots, P'_{m'}$ are atomic formulas, and $\bar{y}$ and $\bar{y'}$ are sequences of domain variables.[3] *If x_i is dominated by the bottom level of L_{i+1} for $1 \leq i \leq k-1$, and the top level of every L'_i is dominated by x_k for $1 \leq i \leq k'$, then every dynamic inference channel in α is removable.*

For example, there are no dynamic inference channels in constraint (14.1) according to Theorem 14.6. Suppose that we strengthen the requirement by replacing constraint (14.1) with the following labeling constraint (14.6), which states that polyinstantiation is prohibited for the MissionId attribute of MT:

$$(\forall l_1, l_2 \in \mathcal{L})(\forall x, y, z)(\mathrm{MT}^{l_1}(x, y) \wedge \mathrm{MT}^{l_2}(x, z) \rightarrow y = z) \tag{14.6}$$

then there are dynamic inference channels at l for every $l \neq \top$, because the insertion of a low MT tuple having the same MissionId as another high MT tuple would be prohibited. Moreover, not all dynamic inference channels are removable, since the insertion of an MT tuple at m_1 having the same MissionId as another MT tuple at m_2 could not be extended with the deletion of any high MT tuples to avoid dynamic inference channels. However, suppose that we replace constraint (14.1) with the following labeling constraint (14.7) instead, which states that polyinstantiation is prohibited at comparable levels for the MissionId attribute of MT:

$$(\forall l_1 \in \mathcal{L}, l_2 \in \mathcal{L}_{l_1})(\forall x, y, z)(\mathrm{MT}^{l_1}(x, y) \wedge \mathrm{MT}^{l_2}(x, z) \rightarrow y = z) \tag{14.7}$$

then there are still dynamic inference channels at l for every $l \neq \top$, but all of them are removable according to Theorem 14.6. The insertion of a low MT tuple having the same MissionId as another high MT tuple could be extended with the deletion of the high tuple to avoid dynamic inference channels. Notice that for a totally ordered lattice, constraints (14.6) and (14.7) are equivalent.

Not every update policy is acceptable. A natural requirement of an update policy is that the extended multilevel update should not cause a loss of high data. In other words, a low update should only be extended with high insertion. A *lossless* dynamic inference channel is one for which such a lossless update policy exists. For example, the above update policy for constraint (14.7) is not lossless.

Theorem 14.7 *If all relational formulas in labeling constraint α have negative polarity, then no dynamic inference channel in α is lossless.*

According to Theorem 14.7, no dynamic inference channels in constraint (14.7) are lossless. Hence they cannot be removed by any lossless update policy.

[3] We assume that the universal level quantifiers are not vacuous; namely, $k \geq 1$, and every x_i appears in some P_j for $1 \leq i \leq k$ and $1 \leq j \leq m$.

Theorem 14.8 *Suppose that labeling constraint α has the form*

$$(\forall x \in L)(\forall \bar{y})(P_1 \wedge \cdots \wedge P_m \rightarrow (\exists x_1 \in L_1, \ldots, x_k \in L_k)(\exists \bar{y'})(P'_1 \wedge \cdots \wedge P'_{m'}))$$

where $P_1, \ldots, P_m$ are relational formulas, $P'_1, \ldots, P'_{m'}$ are atomic formulas, and $\bar{y}$ and $\bar{y'}$ are sequences of domain variables.[4] *If x is the top level of L_i for $1 \leq i \leq k$, then every dynamic inference channel in α is lossless.*

For example, there are dynamic inference channels in constraint (14.2) at every level l for $l \neq \top$, when a deleted low primary key value is referred to by an existing high foreign key value. However, we can define a lossless update policy that removes all these dynamic inference channels according to Theorem 14.8, because the low deletion of the primary key value could always be extended with a high insertion of the same primary key value.

Not every lossless update policy is acceptable. A natural requirement of a lossless update policy is that the extended multilevel update should be minimized in the amount of change to high data. In other words, the number of high insertions should be minimal. However, a minimal and lossless update policy is in general not unique. For example, a minimal update policy for constraint (14.2) could extend a deletion of the first two MT tuples in Figure 12.1 with the insertion of either tuple at $\top$, both of which are minimal. We could often achieve uniqueness by considering subclasses of updates. For example, if we restrict ourselves to updates consisting of the insertion or deletion of single tuples, then a unique, minimal, and lossless update policy can be defined that removes every dynamic inference channel in constraint (14.2), as is shown in [37].

Theorem 14.9 *Suppose that labeling constraint α has the form*[5]

$$(\forall x \in L)(\forall \bar{y})(P_1 \wedge \cdots \wedge P_m \rightarrow (\exists! x_1 \in L_1, \ldots, x_k \in L_k)(\exists! \bar{y'})(P'_1 \wedge \cdots \wedge P'_{m'}))$$

where $P_1, \ldots, P_m$ are relational formulas, $P'_1, \ldots, P'_{m'}$ are atomic formulas, and $\bar{y}$ and $\bar{y'}$ are sequences of domain variables.[6] *If x is the top level of L_i for $1 \leq i \leq k$, then there is a unique, minimal, and lossless update policy that removes every dynamic inference channel in α.*

For example, if we strengthen constraint (14.2) into the following labeling constraint (14.8), which states that every foreign key value refers to a unique visible primary key value:

$$(\forall l_1 \in \mathcal{L})(\forall x, y, z)(\mathrm{SMD}^{l_1}(x, y, z) \rightarrow (\exists! l_2 \in \mathcal{L}^{l_1})(\exists! w)\mathrm{MT}^{l_2}(y, w)) \tag{14.8}$$

then a unique, minimal, and lossless update policy can be defined that removes every dynamic inference channel according to Theorem 14.9. Since every MT tuple is referred to by at most one SMD tuple, a deleted low MT tuple would have to be re-inserted at high if it is referred to by any high SMD tuple.

When labeling constraints are restricted to polyinstantiation and referential security properties, we could do even better. In fact, the update semantics of Section 13.3 actually defines an update policy for these constraints, which is unique, minimal, and lossless.

[4] We assume that the existential quantification is not vacuous; namely, $m' \geq 1$, and there is an x_i in every P'_j for $1 \leq i \leq k$ and $1 \leq j \leq m'$.

[5] We use $\exists!$ to denote the quantifier "there exists a unique...".

[6] Again, we assume that the existential quantification is not vacuous.

14.7 Design Guidelines

Our results offer valuable guidelines to database designers for the appropriate specification and simple characterization of inference channel-free labeling constraints, both positive and negative. The following are some sample guidelines:

1. The class of single-level labeling constraints, such as the polyinstantiation security property, is a safe class of labeling constraints, since it is free of both static and dynamic inference channels.

2. If a non-single-level labeling constraint is free of static inference channels, then it is most likely not free of dynamic inference channels, and vice versa, due to the symmetry discussed in earlier.

3. The referential security property is free of static inference channels, and it could not be further relaxed without introducing static inference channels.

4. Static inference channels could be removed by enforcing labeling constraints at every level, and dynamic inference channels could often be removed by a proper update semantics.

5. Replacing the polyinstantiation security property with unconditional no-polyinstantiation constraints introduces inference channels that could not be completely removed, even by giving up high data.

6. The referential security property contains lossless dynamic inference channels. However, either updates should be restricted or a stronger form of the property should be enforced in order to achieve a unique, minimal, and lossless update semantics.

Chapter 15

Secure Interoperation

Recent advances in distributed systems and networking technology have made interoperation not only feasible but also increasingly popular. For example, heterogeneous databases can be linked by high-speed networks that consist of heterogeneous networks connected by gateways. In such an application environment, heterogeneity (such as in data semantics, data representation, and communication protocol) among system components must be reconciled properly. Some research efforts are under way to deal with these problems [51].

One attribute of interoperation that needs reconciliation but has not been closely studied is security with regard to access control. Consider an application involving multiple systems dealing with commerce (e.g., national credit databases), finance (e.g., stock market information systems), medicine (e.g., patient records), and defense, each having a distinct access control structure. To facilitate information exchange among such systems, some mapping between the heterogeneous security attributes must be introduced, for example, by the system administrators. Current practices show that these mappings, even if chosen carefully, can result in security breaches that previously did not exist in any individual system (e.g., [58, 33]).

Secure interoperation is a serious concern for military systems[1] as well as commercial ones. For example, consider the information system of a major research organization where Alice, being a project supervisor, is allowed access to Bob's files, but not vice versa. Suppose that this organization has just been purchased by a corporation where Charles is Vice President for research and Diana, being his secretary, has access to his files. After the merger, it seems natural to permit Charles to access Alice's project papers. But if Bob should be allowed access to Diana's file cabinet, there would be a security violation because now Bob would potentially have access (indirectly via Diana and Charles) to Alice's files to which he should be denied access.

Although the security violation in this example may not be too difficult to remove, a real-world system could have hundreds or thousands of entries in its access control list so that choosing a secure yet satisfactory (e.g., with maximum data sharing) mapping between many such access control lists is a daunting task. In other words, interoperation of systems with heterogeneous access control

[1] It is estimated in the Defense Information Systems Agency's *Defense Information System Network Technology Requirements Document*, version 0 (August 3, 1993) that the U.S. DoD enterprise has more than 10,000 networks worldwide, most of which are not interoperable with each other and do not adequately support information sharing.

structures poses the following new challenges: what is the definition of secure interoperation? How can security violations be detected? And how can these violations be removed while a maximum amount of information exchange is still facilitated? We attempts to answer some of these questions. First we turn to what we think are the fundamental requirements in secure interoperation.

15.1 Principles of Secure Interoperation

One essential feature in federated systems is the autonomy of an individual system—each system may be administrated independently [6, 51]. To preserve this feature in secure interoperation, autonomy in security must be guaranteed.

> **Principle of Autonomy.** Any access permitted within an individual system must also be permitted under secure interoperation.

On the other hand, interoperation should not violate the security of an individual system.

> **Principle of Security.** Any access not permitted within an individual system must be also denied under secure interoperation.

All other new access introduced by interoperation should be permitted unless explicitly denied by the specification of secure interoperation. Note that, unless specified otherwise, by access we mean direct or indirect access.

It is conceivable that under some circumstances a system may be willing to sacrifice some of its autonomy.

15.2 System Model and Terminology

In our discussion, the security attributes of a system are expressed with an access control list (ACL) [26]. We view a system as a collection of users, machines, data objects, and others, each being a distinct unit with regard to security.

The task we are facing is the following: given a set of access control lists that are individually secure, define what secure interoperation is, and investigate the complexity of detecting security violations in the global system and that of removing security violations while maintaining a reasonable level of interoperation.

It has been previously shown that the security of any given access control list is in general undecidable [19], and some variations of the decision problem are at best NP-complete [45]. Therefore, we also expect to obtain NP-completeness results and thus follow the general proof method for NP-completeness to investigate only a restricted problem where in each ACL: (1) each subject owns exactly one file, with read and write access; (2) a subject can have only read access to a file owned by someone else; (3) if a subject can read another's file, the latter cannot read the former's file; (4) an ACL is static in that read and write are the only types of access specified.

Our NP-completeness results should imply similar NP-completeness results for formations of the problem using more general access control lists. In addition, given the particular restrictions on

ACL, our results should also imply NP-completeness results for the interoperation of Bell-LaPadula (e.g., [27]) type multilevel secure systems.

In our discussion, we use the following terminology, notations, and definitions. Because one subject owns exactly one file, there is no need to distinguish between a subject and its file. For example, instead of saying that Alice has access to Bob's file, we can simply say that Alice has access to Bob. We call this combination of a subject and its file an entity. Moreover, it is obvious that one entity has access to oneself (i.e., one's own file), and if Alice can access Bob, and Bob can access Charles, then Alice can access Charles indirectly. Recall that one restriction on the ACL is that if Alice can access Bob then Bob cannot access Alice, we arrive at the following definition of a secure system as specified with a restricted ACL.

A *secure system* is an ACL in the form of $G =< V, A >$ where V is a set of entities and A is a binary relation "access" on V that is reflexive, transitive, and antisymmetric.

Graphically, we can view a system as an acyclic directed graph. V is the set of vertices and A is the set of arcs—there is an arc leading from vertex u to v, denoted by (u, v), if and only if A contains the binary relation "u access v". The direction of the arc is then the direction of the permitted "access".

For the merger example, we have that $Res =< \{Alice, Bob, Eve\}, \{(Alice, Bob), (Eve, Alice)\} >$ and $Com =< \{Charles, Diana, Fred\}, \{(Charles, Fred), (Diana, Charles)\}$. The graphical representation of both systems is in Figure 15.1.

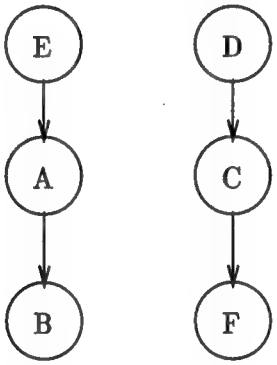


Figure 15.1: Two Separate Systems

For convenience, we sometimes do not distinguish between an ACL and its graphical representation if no confusion can arise.

We say that an access (u, v) is *legal* in G (or in A) iff there is a directed path in (the graphical representation of) G leading from u to v. We denote this with $(u, v) \propto G$.

Suppose we have n secure systems, $G_i =< V_i, A_i >$, $i = 1, 2, \ldots, n$, and for simplicity, we assume that all entities are distinctly named—that is, $V_i \cap V_j = \emptyset, i \neq j$. To facilitate interoperation, mappings between entities of different systems must be introduced to reflect the desired data

sharing through interoperation. Such mappings can be represented by a set of cross-system "access" relations F, which is chosen possibly by an administrator with global security responsibility or by a select committee in charge of the individual systems. *Permitted access* is a binary relation F on $\cup_{i=1}^{n} V_i$ where $\forall (u,v) \in F$, $u \in V_i$, $v \in V_j$, and $i \neq j$. The fact that $(u,v) \in F$ indicates that it is thought that entity u (in system G_i) should be allowed to access entity v (in system G_j). Note that it is possible to have both $(u,v) \in F$ and $(v,u) \in F$.

In our example, suppose that it is decided that interoperation should allow Bob to access Fred (i.e., his file) and Charles to access Alice. Then the global system is in Figure 15.2 where arcs belonging to F are represented as dotted lines.

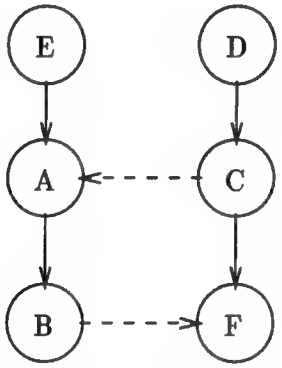


Figure 15.2: Interoperation of Two Systems

The interoperation may also mandate a set of *restricted access* R, which is a binary relation R on $\cup_{i=1}^{n} V_i$ such that $\forall u, v \in R$, $u \in V_i$, $v \in V_j$, and $i \neq j$. This is similar to a negative entry in an access control list [46]. The purpose is to explicitly safeguard certain parts of the system when the potential implications of introducing F are unclear. In our example, we may forbid access $(Diana, Eve)$. R takes precedence over F.

To define secure interoperation for a federated system $Q =< W, B >$, recall that the autonomy principle requires that a legal access in A_i remain legal in B, i.e., if $(u,v) \propto A_i$ then $(u,v) \propto B$. On the other hand, the security principle requires that an illegal access in A_i remain illegal in the interoperation, i.e., if $(u,v) \not\propto A_i$ then $(u,v) \not\propto B$. In addition, all access in R should be explicitly restricted—that is, $B \cap R = \emptyset$ (the empty set). Q is a *secure interoperation* if $B \cap R = \emptyset$, and $\forall u, v \in V_i$, $(u,v) \propto A_i$ if and only if $(u,v) \propto B$.

F and R may contradict each other, and other security violations can also occur as a result of interoperation. As illustrated in Figure 15.3, Bob can access Alice indirectly through Diana, which is illegal within the research organization.

In situations like this, F may need to be changed or reduced to remove security violations (recall that R takes precedence over F). Thus, given $G_i, i = 1, \ldots, n$, F, and R, our aim is to find a federated system $Q =< W, B >$, where $W = \cup_{i=1}^{n} V_i$ and $B \subseteq (\cup_{i=1}^{n} A_i \cup F) - R$, such that Q is a

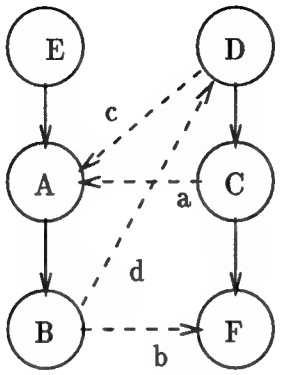


Figure 15.3: Security Violation Caused by Interoperation

secure interoperation.

15.3 Complexity

For convenient discussion, we mark all arcs belonging to $G_i, i = 1, \ldots, n$, green, mark all arcs in the permitted access set F purple, and mark all arcs in the restricted access set R red.

The first problem we encounter is to decide if a given interoperation is secure.

Theorem 15.1 *Security evaluation is in P.*

If $B = (\cup_{i=1}^{n} A_i \cup F) - R$ is insecure, we can remove the security violations by reducing F until the resulting interoperation is secure. In other words, find $S \subseteq F$ such that $C = (\cup_{i=1}^{n} A_i \cup S) - R$ is secure. This is trivial because $S = \emptyset$ is definitely a secure solution.

To find nontrivial secure solutions, one choice is to find a secure solution that includes all other secure solutions. In other words, find $S \subseteq F$ such that $C = (\cup_{i=1}^{n} A_i \cup S) - R$ is secure and, for any secure solution T, $T \subseteq S$. Unfortunately, such solutions do not always exist, as is shown by the following counterexample.

> Consider the interoperation of $G_1 =< \{a1, a2, a3\}, \{(a1, a2), (a2, a3)\} >$ and $G_2 =< \{b1, b2, b3\}, \{(b1, b2), (b2, b3)\} >$, as illustrated by Figure 15.4. Suppose $F = \{(b3, a2), (a3, b2)\}$, which obviously causes a security violation because access $(a3, a2)$ is legal in the federated system but illegal in G_1. One secure solution is $S_1 = \{(a3, b2)\}$. Another secure solution is $S_2 = \{(b3, a2)\}$. But any solution containing both S_1 and S_2 contains F, which causes a security violation.

An alternative in finding nontrivial secure solutions is to look for solutions that cannot be expanded any further. In other words, find a secure solution $S \subseteq F$ such that, for any secure

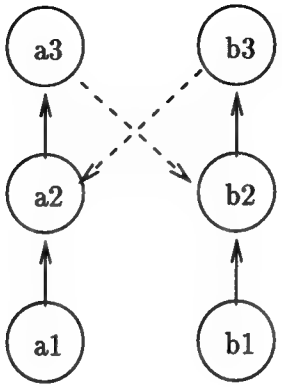


Figure 15.4: All-Inclusive Solutions May Not Exist

solution T, $S \not\subseteq T$. This problem is in P, as the following polynomial-time algorithm demonstrates: start with an empty solution S; add elements in F to S one by one, and only if the addition will not cause a security violation (recall that security evaluation is in P); repeat this process until no more elements can be added. The correctness of this algorithm is obvious.

The three choices described so far do not give natural optimality measures. For example, a solution may turn out to contain just one arc from F although the exclusion of this single arc would allow the addition of two other arcs, with the latter intuitively facilitating more information exchange.

Therefore, we propose two definitions that are more natural. From now on, we stipulate that $F \neq \emptyset$ because the secure interoperation problem disappears when $F = \emptyset$ (and thus $R = \emptyset$).

One natural optimality measure is to maximize direct information sharing. Take the interoperation represented in Figure 15.3, for example. Arcs a and d (or c and d) cause a security violation. To reduce a minimum number of arcs from F, it is better to remove d so that both a and c can be preserved.

Theorem 15.2 *Maximum secure interoperation is NP-complete.*

So far we have been working to find maximum subsets of F that result in secure interoperation, and Theorem 15.2 suggests that this is hard.

Another natural measure of optimality is to maximize direct and indirect information sharing by working on the whole federated system. The aim is to find a secure interoperation with a maximum number of legal access, instead of looking for a secure solution F of a maximum size. That is, we can now change F as long as the new F does not introduce an access that is illegal under the initial set F.

Take the interoperation represented in Figure 15.3 again, for example. Arcs a and d (or c and d) cause a security violation. Previously, for a solution with maximum size, it was better to remove d so that both a and c could be preserved. Now to obtain maximum access, it is actually better to

remove both a and c to preserve d because the latter facilitates more (albeit indirect) information sharing.

Theorem 15.3 *Maximum-access secure interoperation is NP-complete.*

The above results show that the problems we are investigating are NP-complete in general. Nevertheless, we have found a simplified case where each every G_i is a total order.

Theorem 15.4 *Simplified maximum-access secure interoperation is in P.*

The above theorem is very encouraging and more polynomial-time solvable subcases would be desirable.

15.4 Composability

To reduce the total complexity of finding maximum secure interoperation, one area for exploration is the topology of system interoperation. In some federated systems, for example, interoperation is accomplished by having a master system interacting with other systems in local interoperation [51]. We now prove that in such a configuration, the global interoperation is secure if and only if each local interoperation is secure.

Given systems $G_i =< V_i, A_i >, i = 0, 1, \ldots, n$, where G_0 is the master system, let $G_{0,i} =< G_0, G_i, F_i >$ denote the local interoperation between G_0 and G_i with permitted access set $F_i, i = 1, \ldots, n$. The global system is thus $G' =< \cup_{i=0}^{n} V_i, (\cup_{i=0}^{n} A_i) \cup (\cup_{i=1}^{n} F_i)$.

Theorem 15.5 *G' is secure if and only if $G_{0,i}$ is secure, $i = 1, \ldots, n$.*

This theorem implies that local secure interoperation, and thus local maximization, can be computed independently and in parallel.

Corollary 15.6 *G' is a maximum secure interoperation if and only if $G_{0,i}$ is a maximum secure interoperation, $i = 1, \ldots, n$.*

The two very positive results above indicate that in a star-like configuration, global (maximum) secure interoperation can be achieved in a distributed fashion, locally, and *incrementally* as more systems join the interoperation. We can thus say that (maximum) secure interoperation is *composable*. Note that these results do not necessarily imply that maximum-access secure interoperation is composable.

The proofs in Theorem 15.5 clearly extend to any configuration of a tree structure in that if all local interoperation between neighboring systems are secure or maximum, then the global interoperation is also secure or maximum.

Corollary 15.7 *Secure interoperation and maximum secure interoperation are composable in any tree-structure configuration.*

In a ring-structure configuration (or any configuration containing a ring), the composability theorem does not always hold. A simple counterexample is when each F_i contains only one arc; thus, each local interoperation is secure, but the collection of these plus a green arc forms a cycle and permits an illegal access. The implication is that secure interoperation can be joined together as long as no ring is formed.

From the proof details, we expect that the above composability results generalize beyond the simple access control structure we have assumed in our current discussion.

Chapter 16

Conclusion

We have developed a formal policy framework of MAC policies in multilevel relational databases. We have identified seven important components of such policies, and have characterized their desirable properties.

Besides the four components in the traditional interpretation of MAC policies in multilevel databases, one of the most important new components is the interpretation policy. By mapping multilevel relational databases to multilevel theories and structures, the superficial syntactic difference in object labels is abstracted away, and the semantic difference hidden in object labels is made precise. As a consequence, the interpretation policy makes it possible to compare the semantics of multiple MAC policies. As examples, we have developed natural interpretation policies for multilevel relational databases with tuple-level and element-level labeling respectively, which have properties that are commonly recognized as desirable. Based on these policies, we have provided practical design trade-offs in choosing between tuple-level and element-level labeling.

The second new component, the view policy, specifies the upward information flow requirements for a set of view constraints. A view policy should have three desirable properties:

1. it ensures the validity of view constraints,

2. it maximizes upward information flow, and

3. it is deterministic.

As an example, we have developed a view policy for multilevel relational databases with tuple-level labeling, where the view constraints consist of key-based functional and referential dependencies, which has all the desirable properties identified above.

The third new component, the update policy, specifies the mechanisms to eliminate inference channels in the enforcement of a set of labeling constraints. An update policy should also have three desirable properties:

1. it does not introduce inference channels,

2. it does not affect data at lower or incomparable levels, and

3. it does not cause data loss at higher levels.

Based on these properties, we have provided practical design guidelines for the appropriate specification of labeling constraints, whose enforcement would not jeopardize secrecy requirements. As an example, we have developed an update policy for multilevel relational databases with tuple-level labeling where the labeling constraints consist of polyinstantiation and referential security properties, which has all the desirable properties identified above.

Based on the framework, we have compared the MAC policies commonly imposed in or proposed for multilevel relational databases. Our framework could be used to capture and resolve the MAC policy mismatches in the secure interoperation of heterogeneous multilevel databases. As an initial step in this direction, we have investigated the secure interoperation of multilevel databases whose MAC policies mismatch in the lattice component.

Part III

Appendix: Prototype System Design and Implementation

Chapter 17

Introduction

We have developed a query mediation concept demonstration prototype. This appendix focuses on the prototype itself. First a short general description of the prototype's functionality is provided in Chapter 18. Next, some information about the implementation is given in Chapter 19. Finally, the demonstration is described in detail. A high-level description of each of the four examples that constitiute the demonstration can be found in Chapter 20, followed by an annotated transcript of the demonstration that fills in the low-level details in Chapter 22.

Chapter 18

Prototype Functionality

The mediator prototype performs three principal functions. First, it translates queries expressed in database query languages into the mediator's logic-based internal representation, and, conversely, translates queries expressed in the internal representation into database query languages. Second, it transforms a query on one database into a collection of queries on other databases that, when evaluated, will provide data relevant to the original query. Third, it translates the tables produced by evaluating the collection of queries into a table of responses to the original query.

18.1 Query Translation

The present prototype translates two dialects of SQL, Standard SQL and the version of SQL used by Oracle.[1] into the internal representation language. (Only queries of the form

SELECT *select-clause-list**
FROM *relation-list**
WHERE *conjoined-equations-and-inequalities**;

are translated by the prototype, because the query transformation function implemented in the prototype can only handle queries of this form.) The query is first parsed, producing an abstract syntax tree, and then additional syntactic checking is performed, using the mediator's internal representation of the originating database's schema.

Prior to the translation into logic, the query is *minimized.* Minimization is a way of eliminating inessential dependencies built into the schema. For example, suppose that the originating database contains a relation R with attributes A_0, A_1, and A_2 where A_0 is a primary key. The relation R could be broken down into three relations

$$R_0 = \{\{A_0 \mapsto x_0\} : \text{ for some } x_1 \text{ and } x_2, \{A_0 \mapsto x_0, A_1 \mapsto x_1, A_2 \mapsto x_2\} \in R\}$$
$$R_1 = \{\{A_0 \mapsto x_0, A_1 \mapsto x_1\} : \text{ for some } x_2, \{A_0 \mapsto x_0, A_1 \mapsto x_1, A_2 \mapsto x_2\} \in R\}$$
$$R_2 = \{\{A_0 \mapsto x_0, A_2 \mapsto x_2\} : \text{ for some } x_1, \{A_0 \mapsto x_0, A_1 \mapsto x_1, A_2 \mapsto x_2\} \in R\}$$

[1] All product and company names mentioned in this report are trademarks of their respective holders.

without loss of information. If a query Q that involves R can be rewritten as a query involving only one of R_1 and R_2, the likelihood of finding information relevant to Q in another database is greatly increased. For example, a target database might well contain a relation R_1' with tuples semantically relevant to R_1 but no information semantically relevant to R_2. (An additional advantage is that decomposing queries into multiple queries over different databases becomes easier.) Minimization consists of rewriting the query to use such minimal—i.e., minimal relative to the semantic constraints imposed by the keys—relations. For the relation R above, this would be done by rewriting references to attributes

$$R.A_i \longrightarrow R_i.A_i \qquad (i \in \{0,1,2\})$$

and adding the constraint

$$R_0.A_0{=}R_i.A_0$$

to the `WHERE` clause if R_i is mentioned ($i \in \{1,2\}$), so that joins are performed when necessary.

The process of translating a minimized query into the logical language is conceptually straightforward: the `FROM` and `WHERE` clauses of the query supply the matrix of the formula, and then the variables in the matrix are existentially quantified unless the corresponding attributes are mentioned in the `SELECT` clause. For example, the query

```
SELECT  R.A
FROM    R, S
WHERE   R.B=S.C;
```

where the scheme of R is $\{A, B\}$ and the scheme of S is $\{C, D\}$, would be translated to

$$\exists x_{\mathrm{B}}\, \exists x_{\mathrm{C}}\, \exists x_{\mathrm{D}}\, [\, R(x_{\mathrm{A}}, x_{\mathrm{B}}) \wedge S(x_{\mathrm{C}}, x_{\mathrm{D}}) \wedge x_{\mathrm{B}} = x_{\mathrm{C}}\,]$$

18.2 Query Transformation

After translation, logical simplification is performed. Minimization often introduces considerable redundancy, which can profitably be eliminated prior to attempts to transform the query. In terms of the earlier example, where relation R is replaced by relations R_0, R_1 and R_2, the mediator can use the fact that R_0 is implied by R_i ($i \in \{1,2\}$) to simplify a formula such as

$$R_0(x) \wedge R_i(y,z) \wedge x = y \wedge x < k$$

to

$$R_i(y,z) \wedge y < k$$

Simplification rules for eliminating redundancy due to minimization are generated automatically. If a database scheme contains additional semantic redundancy, which is not uncommon in practice, additional rules can be added to the simplifier. Generally, if, for some formulas ϕ and ψ, the simplifier is told that ϕ implies ψ, then it will simplify

$$\phi \wedge \psi[x/x', y/y', \ldots, z/z'] \wedge x = x' \wedge y = y' \wedge \cdots \wedge z = z' \wedge \bigwedge \Gamma$$

to

$$\phi \wedge \bigwedge \Gamma[x'/x, y'/y, \ldots, z'/z]$$

After simplification, rewrite rules are applied in an attempt to replace the vocabulary of the originating database with the vocabulary of some target database. A rewrite rule either replaces a term by a term or replaces a conjunction of literals—which can be thought of as a complex predicate—by a conjunction of literals.[2] Examples of typical transformation rules can be found in the discussion of the examples in Chapter 20. The most complex and interesting rule appears in the fourth example. It has the form

$$x = k \longrightarrow x = \alpha(k)$$

where α is code that runs a query involving the constant term k on some database and extracts a constant term, represented here by $\alpha(k)$, from the response. Effectively, the rule says to replace k by some other term $\alpha(k)$ determined by consulting some database.

After all the rules for transforming queries on the originating database to the target database have been applied, either all the vocabulary of the originating database has been replaced by the vocabulary of the target database, or some remains. If all the originating vocabulary has been replaced, then we have a query on the target database, which is then translated to (the appropriate version of) SQL, "de-minimized", and executed to obtain additional data relevant to the original query. If some of the originating vocabulary remains, we have a query that cannot be executed on the target database. In this case, we have two alternatives. First, we can simply abandon the attempt to retrieve relevant data from the target. In the prototype, this alternative is chosen when the attempt to rewrite the query has no effect on it, that is, when none of the rules were applicable. Second, we can attempt to build on partial success by applying the rules for transforming queries on the originating database to some other target database to the partially transformed query. If these rules eliminate the remnants of the original vocabulary, the result is a query over the combination of the two databases.

In theory, this process could be repeated, resulting in a query over some combination of a large number of target databases. In practice, the combinatorics prohibit complex combination; even exhaustive exploration of all pairs of targets is too expensive, given the small probability that any given pair will yield useful information. Therefore, the mediator contains knowledge of which other targets should be considered in the case of partial success as part of its control strategy.

The query over the combined databases must then be broken down into queries over the several individual databases, plus "glue" for combining the results of executing those queries. Sorting the query's conjuncts is straightforward. A literal involving a relation on one of the databases becomes part of the query on that database. If an equation or inequality is between two terms associated with a database, it becomes part of the query on that database. Otherwise, it relates terms across databases and is reserved for "gluing" the results together. See the third example in Chapter 20 for a case where an equation that corresponded to an equijoin in the first example is used as "glue".

[2] Note that the formula produced by translating the query is an existentially quantified conjunction of literals. Even when more complex `WHERE` clauses are eventually supported, the matrix of the formula will still be in disjunctive normal form, and so the present rewrite rules can still be applied disjunct by disjunct.

The correctness criterion for rules is that the formula on the right-hand-side should imply, given the semantics of the relations and terms, the formula on the left-hand-side. This means that any sequence that satisfies the transformed query will satisfy the original, relative to the intended semantics. Therefore, the tuples returned by executing the transformed query are relevant to the original query, in the sense that the user has asked for all data satisfying a certain semantic property, and these data have that property.

18.3 Table Translation

If query transformation produces a single query, table translation is simply a matter of applying the term rewriting rules "backward", to replace the terms of the target database by the terms of the originating database, and reordering the columns (if necessary). The resulting table looks like a response to the original query, which can be presented to the user as additional data. If the transformation produces multiple queries and "glue", the resulting tables must first be combined using that glue. The third example in Chapter 20 illustrates using an equation to combine tables, that is, to perform an equijoin.

Chapter 19

Prototype Software

The mediator runs as a Lisp process under Unix, communicating with database managements systems (DBMSs) and DBMS user interfaces via ASCII files. For example, in the demonstration, a wrapper around a database system writes the user's query to a file. The mediator uses the file name to determine which wrapper sent the query, i.e., which database the query is on. It reads the query, and writes any resulting queries to files that are monitored by the various DBMS wrappers. The same process is used to pass tables between wrappers and the mediator. Periodic polling of ASCII files is used rather than some more direct form of Unix interprocess communication in order to achieve greater operating system independence. Any system that can exchange files with the system running the mediator can host a mediated database.

The mediator code resides in a single Lisp package, called `MERRIMACK`. The source code is organized as a collection of files corresponding to data structures used by the mediator (e.g., `formulas.lisp`, which defines the data structures that implement logical formulas) and functions performed by the mediator (e.g., `parse-sql.lisp`, which contains the code for parsing queries), together with a system definition file `merrimack.lisp` which defines the `MERRIMACK` package and loads the other files. Code specific to the demonstration—database schema definitions, the particular transformation rules used, etc.—is in the file `demo.lisp`. An image containing the code for the demonstration is just under 7.8 Mbytes.

Chapter 20

Examples

The demonstration consists of a series of four examples. The four can be run any number of times, in any order, but are naturally ordered by the complexity of the functions performed by the mediator. Each example is briefly described below; the details can be found in Chapter 22, which is an annotated transcript of the output produced by the mediator during a demonstration.

20.1 First Example: Basic Query Transformation

Imagine that a physician working at a clinic has recently observed an unexpected allergic reaction of a patient to an experimental drug, XD2001. As a result, she decides to search her local database for information about other recent allergic reactions to that drug. The schema for the relevant part of her database[1] is

PATIENTS	PATIENT_ID	TRANSACTION_TIME

PATIENT_ALLERGY	PATIENT_ID	DRUG_NAME	NOTE_ID	START_TIME	...

NOTES	NOTE_ID	TEXT

Thus, the appropriate SQL query on this database is

```
SELECT PATIENT_ALLERGY.PATIENT_ID, TEXT
FROM   PATIENT_ALLERGY, PATIENTS, NOTES
WHERE  PATIENT_ALLERGY.PATIENT_ID = PATIENTS.PATIENT_ID
AND    PATIENT_ALLERGY.NOTE_ID = NOTES.NOTE_ID
AND    DRUG_NAME = 'XD2001'
AND    '1-JAN-94' ¡ TRANSACTION_TIME;
```

Unknown to our physician, data relevant to her query are stored in the database of a local hospital that is joined to the clinic via the mediator. Of course, there are differences in the way that the information is represented, which means that the same query cannot be used to retrieve it. The hospital's schema is

[1] These relations schemes were extracted from the THelper-II database of Stanford Medical School.

ADMISSIONS	PATIENT_ID	ADMISSION_TIME	PATIENT_NAME	···

DRUG_ALLERGY	PATIENT_ID	DRUG_ID	TEXT

Three differences between the two databases are important to this example.

1. The hospital uses a different name for the drug, the tradename "Druggo" rather than the scientific designation "XD2001".

2. Notes about allergic reactions are stored in the relation that records whether a reaction has occurred in the hospital database, while all notes are recorded in a single relation in the clinic database (presumably to save space).

3. Most importantly, the hospital database stores less precise information about the time of the treatment that caused the allergic reaction than the clinic database does. While the clinic database stores the transaction time associated with the treatment, the hospital database stores only the admission time and release time for the patient.

So, to transform this query, the mediator needs rules that

1. transform `XD2001` to `DRUGGO`,

2. transform a join of `PATIENT_ALLERGY` and `NOTES` to `DRUG_ALLERGY`, and

3. transform a request for information with a `TRANSACTION_TIME` later than some given time to a request for information with an `ADMISSION_TIME` later than that given time (since any transaction at the hospital must occur after admission).

This example demonstrates the query rewriting process in which the mediator resolves the semantic and representational mismatches between the clinic database and the hospital database, using its knowledge about the relationships between the two databases. As the result, our physician at the clinic is able to access relevant data in both databases without even knowing the existence, schema, and semantics of the hospital database.

20.2 Second Example: Reversed Roles

The second example is similar to the first in terms of requirements on the mediator's functionality, but it makes the point that the connection through the mediator is symmetric. This time, a physician at the hospital wants to determine whether a patient, whose ID is 123-45-6789, has had a platelet count performed. So he issues the query

```
SELECT RUN_DATE
FROM   TEST_ORDERED
WHERE  PATIENT_ID = '123-45-6789'
AND    ORDERABLE_TEST_NAME = 'PLATELET COUNT';
```

on the hospital database, where the relevant part of the database's schema is

TEST_ORDERED	PATIENT_ID	ORDERABLE_TEST_NAME	RUN_DATE	···

It turns out that the patient has had a platelet count performed at the clinic, where the relevant part of the schema is

LAB_RESULT	PATIENT_ID	TEST_NUMBER	···

TESTS	TEST_NUMBER	TEST_NAME	···

Again, the organization of the databases is somewhat different—e.g., a join of clinic relations LAB_RESULT and TESTS must be performed to determine whether a test with a given name has been run on a given patient, while the hospital stores the information in the single relation TEST_ORDERED—so the query must be appropriately transformed.

This example demonstrates that the mediator is able to perform query rewriting in both directions. Either the clinic database or the hospital database can serve as an entry point for users to issue queries, and users can access data in both databases by only knowing the existence, schema, and semantics of one database. In contrast, in the federated database approach, only the federated schema can serve as the entry point, and users have to understand the federated schema and its semantics in order to gain access to multiple databases. As the result, our physician at the hospital can access data in the clinic database just as easily as physicians at the clinic accessing data in the hospital database.

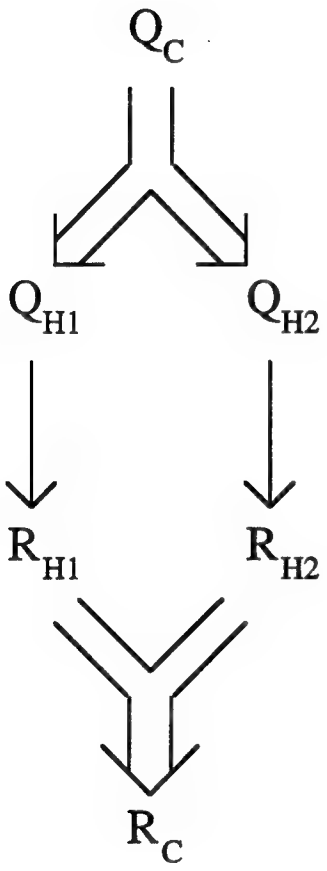
20.3 Third Example: Split Query, Join Tables

This example is, externally, quite similar to the first. Exactly the same query is processed on exactly the same clinic database. The only difference on the hospital side is that the relations ADMISSIONS and DRUG_ALLERGY are now stored in different databases, so that the information we need cannot be retrieved using a single query. Instead, two queries must be generated, and the tables that result from running them combined by the mediator prior to presentation of the information to the issuer of the original query. The processing steps are illustrated graphically in Figure 20.1.

This example demonstrates the ability of the mediator to rewrite a query in one database to a query involving multiple databases. Data in multiple databases are combined to give answer to the original query. As the result, our physician at the clinic gains access to relevant hospital data without knowing which hospital databases contain relevant data and how to combine data from them in a semantically meaningful way.

20.4 Fourth Example: Auxiliary Queries

In the final example, we make a change to the clinic database: rather than using the patient's social security number (SSN) as an ID, a specially generated identifier is used. The correspondence between these identifiers and personal data about the patient, including the patient's SSN, is stored in another database that is accessible only to authorized users. The schema for this restricted-access database is



Query on Clinic Database

transformed by Mediator

Queries on Hospital Databases

run by Hospital DBMSs

Replies to Queries on Hospital Databases

transformed by Mediator

Additional Data Relevant to Original Query

Figure 20.1: Example Three Processing

PERSONAL_DATA	PATIENT_ID	PATIENT_NAME	SSN	···

Our physician is now interested in allergic reactions experienced by one of her patients, who is identified as P11111 in the clinic database where allergy information is stored. She therefore issues the query

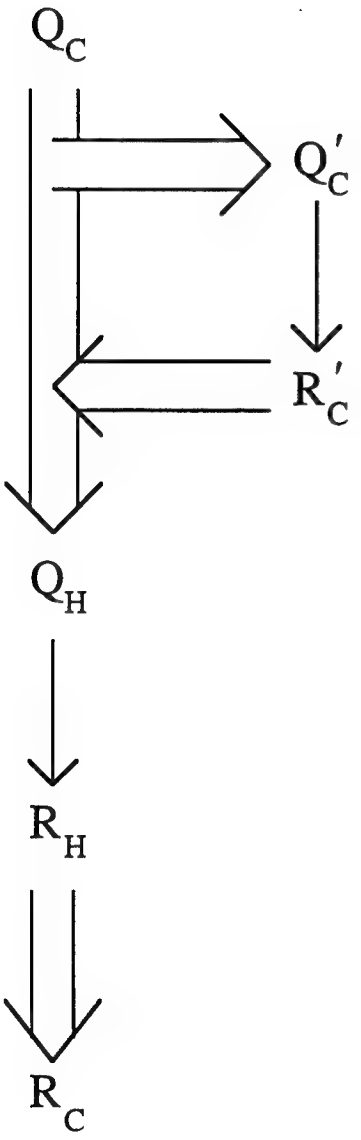
```
SELECT DRUG_NAME, TEXT
FROM   PATIENT_ALLERGY, NOTES
WHERE  PATIENT_ALLERGY.PATIENT_ID = 'P11111'
AND    PATIENT_ALLERGY.NOTE_ID = NOTES.NOTE_ID;
```

Just as in the first example, there are relevant data in the hospital database, but an additional difference between the way the two databases store information is reflected in the patient IDs. In order to generate an equivalent query on the hospital database, the mediator must determine P11111's SSN by running the query

```
SELECT SSN
FROM   PERSONAL_DATA
WHERE  PATIENT_ID = 'P11111';
```

on the restricted-access database—provided, of course, that our physician is allowed access to that information—and extracting the SSN from the response. These processing steps are illustrated graphically in Figure 20.2.

This example demonstrates the capability of the mediator to perform mediation-based access control. The clinic database is unclassified, the restricted-access database is secret, while the hospital database is multilevel where SSN is secret and other data elements are unclassified. Without the mediator, our physician at the clinic cannot access relevant data in the hospital database, even though they are unclassified. Through the mediator, such access is made possible without compromising the secret association between SSN and ID stored in the restricted-access database or the secret SSN stored in the hospital database.



Query on Clinic Database

Query on Restricted-Access Clinic Database
(*generated during transformation*)

Reply to Query on Restricted-Access Clinic Database
(*used to generate query on Hospital Database*)

Query on Hospital Database

Reply to Query on Hospital Database

Additional Data Relevant to Original Query

Figure 20.2: Example Four Processing

Chapter 21

Lessons Learned in Prototype Development

With regard to functionality, the principal lesson learned was that this basic approach to query mediation appears feasible. Although only a subset of SQL's SELECT queries are handled by the prototype's translation and transformation functions, both the query translator and the query transformer can be straightforwardly extended to deal with more complex queries.

Many rule-based systems do not scale well due to exponential growth in the amount of knowledge required to perform effectively in less restricted domains. There is good reason to believe that the mediator prototype will scale relatively well. The query transformation rules encode knowledge about semantic relationships between database schemas in a query language-independent form. As a result, handling larger subsets of SQL will not require any additional transformation rules, because queries will still be translated into the same simple logical representation prior to transformation. More importantly, there is no sense in which the set of transformations must be "complete" in order for the prototype to perform effectively. Each transformation represents an observed semantic relationship between databases. If some semantic relationships are not represented, then some relevant data will not be returned in response to some queries. But whatever data *are* returned *will* be relevant; missing transformations cannot lead to incorrect results. Even a small effort devoted to encoding semantic relationships can yield significant returns, and the resulting rule set can be incrementally extended if the additional benefits seem to warrant the cost.

Lisp proved to be a good choice for the initial prototyping effort. Its advanced facilities for symbolic programming simplified writing many of the functions required for mediation, such as the pattern matcher used in query transformation. An additional advantage of using Lisp is that its dynamic nature makes generation and execution of code at query-transformation time easy. Performance of the prototype was not an issue: query translation and transformation had to be artificially slowed down for purposes of the demonstration, and the mediator's space requirements are quite modest by modern standards. Use of file-based communication between the mediator and the DBMSs was a reasonable first approximation to the sort of asynchronous file-based communication over the Internet used by applications such as Mosaic and Netscape.

Finally, the principal lesson learned from the examples was that writing transformations that

express semantic relationships between actual schemas is quite straightforward. This was most graphically illustrated by the development of the example in Section 20.2, where the roles of the databases were reversed. The very first attempt to write a rule relating test data representation in the hospital database to test data representation in the clinic database—simply a matter of breaking down a complex relation into simpler relations, a very typical sort of difference in data representation—was sufficient to handle this example. Although our experience to date is limited to having written a few dozen rules relating a small number of database schemas, it strongly suggests that expressing observed semantic relationships in our formalism will not be difficult.

Chapter 22

Transcript of Demonstration

What follows is a transcript of the output produced by the mediator in a demonstration run. Text produced by the mediator is in `teletype` font. Text entered by the person running the mediator is in *italic teletype*. Annotations added to the transcript are in *roman italics*.

Start the mediator by typing its name at the Unix prompt.

```
% merrimack
Starting Mediator ...
```

The mediator is waiting for input. This is provided by creating a file that contains the query to be processed. The name of the file tells the mediator where the query originated and what other databases are candidates for providing additional data.

```
Reading SQL from file "demo-in.sql" ...
```

"demo-in.sql" indicates that the query is from the Clinic database and that the Hospital database is a candidate.

```
Abstract syntax tree for the SQL form is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY PATIENT_ID)
                    (:alias NIL))
      (:select-item (:attribute NIL TEXT)
                    (:alias NIL)))
   (:relations
      (:relation PATIENT_ALLERGY)
      (:relation PATIENTS)
      (:relation NOTES))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
```

```
         (:attribute PATIENTS PATIENT_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY NOTE_ID)
         (:attribute NOTES NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute NIL DRUG_NAME)
         (:literal-value XD2001))
      (:constraint
         (:predicate >)
         (:attribute NIL TRANSACTION_TIME)
         (:literal-value 01-JAN-94))))
Continue?  (Y or N):  y
```

When run in interactive mode, the mediator pauses at the end of each processing stage and provides the option of continuing the demo or not.

```
Filling in omitted attributes and aliases in SQL tree ...
Completed abstract syntax tree for the SQL form is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY PATIENT_ID)
                    (:alias PATIENT_ID))
      (:select-item (:attribute NOTES TEXT)
                    (:alias TEXT)))
   (:relations
      (:relation PATIENT_ALLERGY)
      (:relation PATIENTS)
      (:relation NOTES))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
         (:attribute PATIENTS PATIENT_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY NOTE_ID)
         (:attribute NOTES NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY DRUG_NAME)
         (:literal-value XD2001))
      (:constraint
         (:predicate >)
         (:attribute PATIENTS TRANSACTION_TIME)
         (:literal-value 01-JAN-94))))
Continue?  (Y or N):  y
```

```
Converting query to minimized representation ...
Minimized abstract syntax tree is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY PATIENT_ID)
                    (:alias PATIENT_ID))
      (:select-item (:attribute NOTES_TEXT TEXT)
                    (:alias TEXT)))
   (:relations
      (:relation PATIENT_ALLERGY)
      (:relation PATIENTS)
      (:relation NOTES)
      (:relation NOTES_TEXT)
      (:relation PATIENTS_TRANSACTION_TIME))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
         (:attribute PATIENTS PATIENT_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY NOTE_ID)
         (:attribute NOTES NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY DRUG_NAME)
         (:literal-value XD2001))
      (:constraint
         (:predicate >)
         (:attribute PATIENTS_TRANSACTION_TIME TRANSACTION_TIME)
         (:literal-value 01-JAN-94))
      (:constraint
         (:predicate =)
         (:attribute NOTES NOTE_ID)
         (:attribute NOTES_TEXT NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENTS PATIENT_ID)
         (:attribute PATIENTS_TRANSACTION_TIME PATIENT_ID))))
Continue?  (Y or N):  y

Translating SQL query to logical formula ...
Logical form of query is:
(E ?patient_allergy.note_id)
(E ?patient_allergy.drug_name)
(E ?patients_transaction_time.transaction_time)
(E ?notes.note_id)
```

```
(E ?notes_text.note_id)
(E ?patients.patient_id)
(E ?patients_transaction_time.patient_id)
   \/{ /\{  Patient_Allergy(
              ?patient_allergy.patient_id,
              ?patient_allergy.drug_name,
              ?patient_allergy.note_id),
            Patients(?patients.patient_id),
            Notes(?notes.note_id),
            Notes_Text(?notes_text.note_id, ?notes_text.text),
            Patients_Transaction_Time(
              ?patients_transaction_time.patient_id,
              ?patients_transaction_time.transaction_time),
            ?patient_allergy.patient_id = ?patients.patient_id,
            ?patient_allergy.note_id = ?notes.note_id,
            ?patient_allergy.drug_name = xd2001,
            ?patients_transaction_time.transaction_time > 01-jan-94,
            ?notes.note_id = ?notes_text.note_id,
            ?patients.patient_id = ?patients_transaction_time.patient_id}}
where the free variables of the formula are associated with query attribute
aliases as follows:
      ?PATIENT_ALLERGY.PATIENT_ID  <--> PATIENT_ID
      ?NOTES_TEXT.TEXT  <--> TEXT
Continue?  (Y or N):  y

Simplifying logical formula ...
Simplified formula is:
(E ?patients.patient_id)
(E ?patient_allergy.note_id)
(E ?notes.note_id)
(E ?patient_allergy.drug_name)
(E ?patients_transaction_time.transaction_time)
   \/{ /\{  Patient_Allergy(
              ?patient_allergy.patient_id,
              ?patient_allergy.drug_name,
              ?patient_allergy.note_id),
            Notes_Text(?notes.note_id, ?notes_text.text),
            Patients_Transaction_Time(
              ?patients.patient_id,
              ?patients_transaction_time.transaction_time),
            ?patient_allergy.patient_id = ?patients.patient_id,
            ?patient_allergy.note_id = ?notes.note_id,
            ?patient_allergy.drug_name = xd2001,
            ?patients_transaction_time.transaction_time > 01-jan-94}}
Continue?  (Y or N):  y

Attempting to derive queries on remote databases ...
```

```
Attempting to derive query on HOSPITAL_DATABASE ...
Succeeded!
```

The rules that were used in this case are

$Patients(p) \longrightarrow Admissions(p,\ ?admissions.admission\_time)$,

$Patient\_Allergy(p,\ d,\ n_1) \wedge Notes\_Text(n_2,\ x) \wedge n_1 = n_2 \longrightarrow Drug\_Allergy\_Text(p,\ d,\ x)$,

$Patients\_Transaction\_Time(p,\ t_1) \wedge t_2 < t_1 \longrightarrow Admissions(p,\ t_1) \wedge t_2 < t_1$,

and

$xd2001 \longrightarrow druggo$.

```
Logical form of derived query is:
(E ?drug_allergy_text.drug_id)
(E ?admissions.patient_id)
(E ?admissions.admission_time)
   \/{ /\{  ?drug_allergy_text.patient_id = ?admissions.patient_id,
            ?drug_allergy_text.drug_id = druggo,
            Drug_Allergy_Text(
              ?drug_allergy_text.patient_id,
              ?drug_allergy_text.drug_id,
              ?drug_allergy_text.text),
            Admissions(?admissions.patient_id, ?admissions.admission_time),
            ?admissions.admission_time > 01-jan-94}}
where the variables in the two formulas are associated as follows:
      ?PATIENT_ALLERGY.PATIENT_ID  <--> ?DRUG_ALLERGY_TEXT.PATIENT_ID
      ?PATIENT_ALLERGY.DRUG_NAME  <--> ?DRUG_ALLERGY_TEXT.DRUG_ID
      ?NOTES_TEXT.TEXT  <--> ?DRUG_ALLERGY_TEXT.TEXT
      ?PATIENTS.PATIENT_ID  <--> ?ADMISSIONS.PATIENT_ID
      ?PATIENTS_TRANSACTION_TIME.TRANSACTION_TIME
          <--> ?ADMISSIONS.ADMISSION_TIME
Continue?  (Y or N):  y

Translating logical formula to SQL query ...
(:sql-tree
   (:select-items
      (:select-item (:attribute DRUG_ALLERGY_TEXT TEXT)
                    (:alias TEXT))
      (:select-item (:attribute DRUG_ALLERGY_TEXT PATIENT_ID)
                    (:alias PATIENT_ID)))
   (:relations
      (:relation DRUG_ALLERGY_TEXT)
      (:relation ADMISSIONS))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute DRUG_ALLERGY_TEXT PATIENT_ID)
         (:attribute ADMISSIONS PATIENT_ID))
      (:constraint
```

```
         (:predicate =)
         (:attribute DRUG_ALLERGY_TEXT DRUG_ID)
         (:literal-value DRUGGO))
      (:constraint
         (:predicate >)
         (:attribute ADMISSIONS ADMISSION_TIME)
         (:literal-value 01-JAN-94))))
Continue?  (Y or N):  y

Converting from minimized representation to actual representation ...
(:sql-tree
   (:select-items
      (:select-item (:attribute DRUG_ALLERGY TEXT)
                    (:alias TEXT))
      (:select-item (:attribute DRUG_ALLERGY PATIENT_ID)
                    (:alias PATIENT_ID)))
   (:relations
      (:relation DRUG_ALLERGY)
      (:relation ADMISSIONS))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute DRUG_ALLERGY PATIENT_ID)
         (:attribute ADMISSIONS PATIENT_ID))
      (:constraint
         (:predicate =)
         (:attribute DRUG_ALLERGY DRUG_ID)
         (:literal-value DRUGGO))
      (:constraint
         (:predicate >)
         (:attribute ADMISSIONS ADMISSION_TIME)
         (:literal-value 01-JAN-94))))
Continue?  (Y or N):  y


Writing SQL query on remote database to file "demo-out.sql"...
Query has been written to file "demo-out.sql"
```

This query is passed to the Hospital's database system for processing.
The resulting table will be written to the file "demo-in.tbl".

```
Continue?  (Y or N):  y
Converting table in file "demo-in.tbl" for use as response to
  original query
Will write result to "demo-out.tbl" ...
Table read from file "demo-in.tbl", and file deleted
Converted table has been written to file "demo-out.tbl"
```

This completes the first phase of the demonstration. The additional data in the converted table can now be presented to the user who issues the original query.

```
Reading SQL from file "demo-2-in.sql" ...
```

"demo-2-in.sql" indicates a query from the Hospital, and that the Clinic database is a candidate for additional data.

```
Abstract syntax tree for the SQL form is:
(:sql-tree
   (:select-items
      (:select-item (:attribute NIL RUN_DATE)
                    (:alias NIL)))
   (:relations
      (:relation TEST_ORDERED))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute NIL PATIENT_ID)
         (:literal-value 123-45-6789))
      (:constraint
         (:predicate =)
         (:attribute NIL ORDERABLE_TEST_NAME)
         (:literal-value PLATELET COUNT))))
Continue?  (Y or N):  y

Filling in omitted attributes and aliases in SQL tree ...
Completed abstract syntax tree for the SQL form is:
(:sql-tree
   (:select-items
      (:select-item (:attribute TEST_ORDERED RUN_DATE)
                    (:alias RUN_DATE)))
   (:relations
      (:relation TEST_ORDERED))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute TEST_ORDERED PATIENT_ID)
         (:literal-value 123-45-6789))
      (:constraint
         (:predicate =)
         (:attribute TEST_ORDERED ORDERABLE_TEST_NAME)
         (:literal-value PLATELET COUNT))))
Continue?  (Y or N):  y

Converting query to minimized representation ...
Minimized abstract syntax tree is:
```

```
(:sql-tree
   (:select-items
      (:select-item (:attribute TEST_ORDERED_RUN_DATE RUN_DATE)
                    (:alias RUN_DATE)))
   (:relations
      (:relation TEST_ORDERED)
      (:relation TEST_ORDERED_RUN_DATE))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute TEST_ORDERED PATIENT_ID)
         (:literal-value 123-45-6789))
      (:constraint
         (:predicate =)
         (:attribute TEST_ORDERED ORDERABLE_TEST_NAME)
         (:literal-value PLATELET COUNT))
      (:constraint
         (:predicate =)
         (:attribute TEST_ORDERED TEST_NUMBER)
         (:attribute TEST_ORDERED_RUN_DATE TEST_NUMBER))
      (:constraint
         (:predicate =)
         (:attribute TEST_ORDERED PATIENT_ID)
         (:attribute TEST_ORDERED_RUN_DATE PATIENT_ID))
      (:constraint
         (:predicate =)
         (:attribute TEST_ORDERED PHYSICIAN_ID)
         (:attribute TEST_ORDERED_RUN_DATE PHYSICIAN_ID))
      (:constraint
         (:predicate =)
         (:attribute TEST_ORDERED ORDERABLE_TEST_NAME)
         (:attribute TEST_ORDERED_RUN_DATE ORDERABLE_TEST_NAME))))
Continue?  (Y or N):  y

Translating SQL query to logical formula ...
Logical form of query is:
(E ?test_ordered.test_number)
(E ?test_ordered_run_date.test_number)
(E ?test_ordered.patient_id)
(E ?test_ordered_run_date.patient_id)
(E ?test_ordered.physician_id)
(E ?test_ordered_run_date.physician_id)
(E ?test_ordered.orderable_test_name)
(E ?test_ordered_run_date.orderable_test_name)
   \/{ /\{  Test_Ordered(
               ?test_ordered.test_number,
               ?test_ordered.patient_id,
               ?test_ordered.physician_id,
```

```
            ?test_ordered.orderable_test_name),
          Test_Ordered_Run_Date(
            ?test_ordered_run_date.test_number,
            ?test_ordered_run_date.patient_id,
            ?test_ordered_run_date.physician_id,
            ?test_ordered_run_date.orderable_test_name,
            ?test_ordered_run_date.run_date),
          ?test_ordered.patient_id = 123-45-6789,
          ?test_ordered.orderable_test_name = platelet count,
          ?test_ordered.test_number = ?test_ordered_run_date.test_number,
          ?test_ordered.patient_id = ?test_ordered_run_date.patient_id,
          ?test_ordered.physician_id
            = ?test_ordered_run_date.physician_id,
          ?test_ordered.orderable_test_name
            = ?test_ordered_run_date.orderable_test_name}}
where the free variables of the formula are associated with query attribute
aliases as follows:
      ?TEST_ORDERED_RUN_DATE.RUN_DATE  <--> RUN_DATE
Continue?  (Y or N):  y

Simplifying logical formula ...
Simplified formula is:
(E ?test_ordered.test_number)
(E ?test_ordered.physician_id)
(E ?test_ordered.patient_id)
(E ?test_ordered.orderable_test_name)
   \/{ /\{  Test_Ordered_Run_Date(
            ?test_ordered.test_number,
            ?test_ordered.patient_id,
            ?test_ordered.physician_id,
            ?test_ordered.orderable_test_name,
            ?test_ordered_run_date.run_date),
          ?test_ordered.patient_id = 123-45-6789,
          ?test_ordered.orderable_test_name = platelet count}}
Continue?  (Y or N):  y

Attempting to derive queries on remote databases ...

Attempting to derive query on CLINIC_DATABASE ...
Succeeded!
```

The rule that was used in this case is

$$Test\_Ordered\_Run\_Date(n,\ p,\ d,\ o,\ t) \longrightarrow Lab\_Result(p,\ n_1) \wedge\ Tests\_Test\_Name(n_2,\ o) \wedge\ n_1\ =\ n_2.$$

```
Logical form of derived query is:
(E ?lab_result.patient_id)
(E ?tests_test_name.test_name)
```

```
(E ?tests_test_name.test_number)
(E ?lab_result.test_number)
   \/{ /\{  ?lab_result.patient_id = 123-45-6789,
            ?tests_test_name.test_name = platelet count,
            Lab_Result(
              ?lab_result.patient_id,
              ?lab_result.test_number,
              ?lab_result.transaction_time),
            Tests_Test_Name(
              ?tests_test_name.test_number,
              ?tests_test_name.test_name),
            ?tests_test_name.test_number = ?lab_result.test_number}}
where the variables in the two formulas are associated as follows:
      ?TEST_ORDERED.PATIENT_ID  <--> ?LAB_RESULT.PATIENT_ID
      ?LAB_RESULT.TEST_NUMBER  <--> ?LAB_RESULT.TEST_NUMBER
      ?TEST_ORDERED_RUN_DATE.RUN_DATE  <--> ?LAB_RESULT.TRANSACTION_TIME
      ?TESTS.TEST_NUMBER  <--> ?TESTS_TEST_NAME.TEST_NUMBER
      ?TEST_ORDERED.ORDERABLE_TEST_NAME  <--> ?TESTS_TEST_NAME.TEST_NAME
Continue?  (Y or N):  y

Translating logical formula to SQL query ...
(:sql-tree
   (:select-items
      (:select-item (:attribute LAB_RESULT TRANSACTION_TIME)
                    (:alias RUN_DATE)))
   (:relations
      (:relation LAB_RESULT)
      (:relation TESTS_TEST_NAME))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute LAB_RESULT PATIENT_ID)
         (:literal-value 123-45-6789))
      (:constraint
         (:predicate =)
         (:attribute TESTS_TEST_NAME TEST_NAME)
         (:literal-value PLATELET COUNT))
      (:constraint
         (:predicate =)
         (:attribute TESTS_TEST_NAME TEST_NUMBER)
         (:attribute LAB_RESULT TEST_NUMBER))))
Continue?  (Y or N):  y

Converting from minimized representation to actual representation ...
(:sql-tree
   (:select-items
      (:select-item (:attribute LAB_RESULT TRANSACTION_TIME)
                    (:alias RUN_DATE)))
```

```
   (:relations
      (:relation LAB_RESULT)
      (:relation TESTS))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute LAB_RESULT PATIENT_ID)
         (:literal-value 123-45-6789))
      (:constraint
         (:predicate =)
         (:attribute TESTS TEST_NAME)
         (:literal-value PLATELET COUNT))
      (:constraint
         (:predicate =)
         (:attribute TESTS TEST_NUMBER)
         (:attribute LAB_RESULT TEST_NUMBER))))
Continue?  (Y or N):  y


Writing SQL query on remote database to file "demo-2-out.sql"...
Query has been written to file "demo-2-out.sql"

Continue?  (Y or N):  y
Converting table in file "demo-2-in.tbl" for use as response to
  original query
Will write result to "demo-2-out.tbl" ...
Table read from file "demo-2-in.tbl", and file deleted
Converted table has been written to file "demo-2-out.tbl"
```

This completes the second phase of the demonstration.

```
Reading SQL from file "demo-3-in.sql" ...
```

This indicates another query from the Clinic, but, in this case, the candidate sources of data are two databases at the Hospital which jointly contain the same data in the Hospital database of the first example. In fact, the very same query that was used in the first example is reused here.

```
Abstract syntax tree for the SQL form is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY PATIENT_ID)
                    (:alias NIL))
      (:select-item (:attribute NIL TEXT)
                    (:alias NIL)))
   (:relations
      (:relation PATIENT_ALLERGY)
      (:relation PATIENTS)
```

```
      (:relation NOTES))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
         (:attribute PATIENTS PATIENT_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY NOTE_ID)
         (:attribute NOTES NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute NIL DRUG_NAME)
         (:literal-value XD2001))
      (:constraint
         (:predicate <)
         (:literal-value 01-JAN-94)
         (:attribute NIL TRANSACTION_TIME))))
Continue?  (Y or N):  y

Filling in omitted attributes and aliases in SQL tree ...
Completed abstract syntax tree for the SQL form is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY PATIENT_ID)
                    (:alias PATIENT_ID))
      (:select-item (:attribute NOTES TEXT)
                    (:alias TEXT)))
   (:relations
      (:relation PATIENT_ALLERGY)
      (:relation PATIENTS)
      (:relation NOTES))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
         (:attribute PATIENTS PATIENT_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY NOTE_ID)
         (:attribute NOTES NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY DRUG_NAME)
         (:literal-value XD2001))
      (:constraint
         (:predicate <)
         (:literal-value 01-JAN-94)
```

```
        (:attribute PATIENTS TRANSACTION_TIME))))
Continue?  (Y or N):  y

Converting query to minimized representation ...
Minimized abstract syntax tree is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY PATIENT_ID)
                    (:alias PATIENT_ID))
      (:select-item (:attribute NOTES_TEXT TEXT)
                    (:alias TEXT)))
   (:relations
      (:relation PATIENT_ALLERGY)
      (:relation PATIENTS)
      (:relation NOTES)
      (:relation NOTES_TEXT)
      (:relation PATIENTS_TRANSACTION_TIME))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
         (:attribute PATIENTS PATIENT_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY NOTE_ID)
         (:attribute NOTES NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY DRUG_NAME)
         (:literal-value XD2001))
      (:constraint
         (:predicate <)
         (:literal-value 01-JAN-94)
         (:attribute PATIENTS_TRANSACTION_TIME TRANSACTION_TIME))
      (:constraint
         (:predicate =)
         (:attribute NOTES NOTE_ID)
         (:attribute NOTES_TEXT NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENTS PATIENT_ID)
         (:attribute PATIENTS_TRANSACTION_TIME PATIENT_ID))))
Continue?  (Y or N):  y

Translating SQL query to logical formula ...
Logical form of query is:
(E ?patient_allergy.note_id)
(E ?patient_allergy.drug_name)
```

```
(E ?patients_transaction_time.transaction_time)
(E ?notes.note_id)
(E ?notes_text.note_id)
(E ?patients.patient_id)
(E ?patients_transaction_time.patient_id)
   \/{ /\{  Patient_Allergy(
              ?patient_allergy.patient_id,
              ?patient_allergy.drug_name,
              ?patient_allergy.note_id),
            Patients(?patients.patient_id),
            Notes(?notes.note_id),
            Notes_Text(?notes_text.note_id, ?notes_text.text),
            Patients_Transaction_Time(
              ?patients_transaction_time.patient_id,
              ?patients_transaction_time.transaction_time),
            ?patient_allergy.patient_id = ?patients.patient_id,
            ?patient_allergy.note_id = ?notes.note_id,
            ?patient_allergy.drug_name = xd2001,
            01-jan-94 < ?patients_transaction_time.transaction_time,
            ?notes.note_id = ?notes_text.note_id,
            ?patients.patient_id = ?patients_transaction_time.patient_id}}
where the free variables of the formula are associated with query attribute
aliases as follows:
      ?PATIENT_ALLERGY.PATIENT_ID  <--> PATIENT_ID
      ?NOTES_TEXT.TEXT  <--> TEXT
Continue?  (Y or N):  y

Simplifying logical formula ...
Simplified formula is:
(E ?patients.patient_id)
(E ?patient_allergy.note_id)
(E ?notes.note_id)
(E ?patient_allergy.drug_name)
(E ?patients_transaction_time.transaction_time)
   \/{ /\{  Patient_Allergy(
              ?patient_allergy.patient_id,
              ?patient_allergy.drug_name,
              ?patient_allergy.note_id),
            Notes_Text(?notes.note_id, ?notes_text.text),
            Patients_Transaction_Time(
              ?patients.patient_id,
              ?patients_transaction_time.transaction_time),
            ?patient_allergy.patient_id = ?patients.patient_id,
            ?patient_allergy.note_id = ?notes.note_id,
            ?patient_allergy.drug_name = xd2001,
            01-jan-94 < ?patients_transaction_time.transaction_time}}
Continue?  (Y or N):  y
```

```
Attempting to derive queries on remote databases ...
Attempting to derive query on HOSPITAL_DATABASE_1 ...
Partially succeeded.
```

Here is the first difference between the two examples. The subset of the full Hospital database that is called "Hospital database 1" does not contain enough information that any query on it provides data relevant to the original query. Thus, the attempt to transform the query into a query on the partial Hospital database only partially succeeds—some of the relations in the Clinic database are still mentioned. The rules that were used in this case are

$Patients(p) \longrightarrow Admissions(p,\ ?admissions.admission\_time)$,

$Patients\_Transaction\_Time(p,\ t_1) \wedge t_2 < t_1 \longrightarrow Admissions(p,\ t_1) \wedge t_2 < t_1$

and

$xd2001 \longrightarrow druggo$.

```
Logical form of derived query is:
(E ?patient_allergy.note_id)
(E ?notes.note_id)
(E ?patient_allergy.drug_name)
(E ?admissions.patient_id)
(E ?admissions.admission_time)
   \/{ /\{  Patient_Allergy(
              ?patient_allergy.patient_id,
              ?patient_allergy.drug_name,
              ?patient_allergy.note_id),
            Notes_Text(?notes.note_id, ?notes_text.text),
            ?patient_allergy.patient_id = ?admissions.patient_id,
            ?patient_allergy.note_id = ?notes.note_id,
            ?patient_allergy.drug_name = druggo,
            Admissions(?admissions.patient_id, ?admissions.admission_time),
            01-jan-94 < ?admissions.admission_time}}
where the variables in the two formulas are associated as follows:
      ?PATIENTS.PATIENT_ID  <--> ?ADMISSIONS.PATIENT_ID
      ?PATIENTS_TRANSACTION_TIME.TRANSACTION_TIME
          <--> ?ADMISSIONS.ADMISSION_TIME
Continue?  (Y or N):  y

Attempting to derive query on combination of HOSPITAL_DATABASE_1
  and HOSPITAL_DATABASE_2 ...
Succeeded!
```

Applying the rules for transforming a query on the Clinic database into a query on Hospital database 2, the other part of the full Hospital database succeeded in eliminating the remaining relations of the Clinic database. The additional rule that was used in this case is

$Patient\_Allergy(p,\ d,\ n_1) \wedge Notes\_Text(n_2,\ x) \wedge n_1 = n_2 \longrightarrow Drug\_Allergy\_Text(p,\ d,\ x)$.

```
Logical form of derived query is:
(E ?admissions.patient_id)
(E ?admissions.admission_time)
(E ?drug_allergy_text.drug_id)
   \/{ /\{  ?drug_allergy_text.patient_id = ?admissions.patient_id,
            ?drug_allergy_text.drug_id = druggo,
            Admissions(?admissions.patient_id, ?admissions.admission_time),
            01-jan-94 < ?admissions.admission_time,
            Drug_Allergy_Text(
              ?drug_allergy_text.patient_id,
              ?drug_allergy_text.drug_id,
              ?drug_allergy_text.text)}}
where the variables in the two formulas are associated as follows:
      ?PATIENT_ALLERGY.PATIENT_ID  <--> ?DRUG_ALLERGY_TEXT.PATIENT_ID
      ?PATIENT_ALLERGY.DRUG_NAME  <--> ?DRUG_ALLERGY_TEXT.DRUG_ID
      ?NOTES_TEXT.TEXT  <--> ?DRUG_ALLERGY_TEXT.TEXT
Continue?  (Y or N):  y

Splitting logical query over combined databases into multiple queries over
  individual databases plus 'glue'...
The logical formula has been split into the following formulas:

(E ?admissions.admission_time)
   \/{ /\{  Admissions(?admissions.patient_id, ?admissions.admission_time),
            01-jan-94 < ?admissions.admission_time}}

(E ?drug_allergy_text.drug_id)
   \/{ /\{  Drug_Allergy_Text(
              ?drug_allergy_text.patient_id,
              ?drug_allergy_text.drug_id,
              ?drug_allergy_text.text),
            ?drug_allergy_text.drug_id = druggo}}

plus the following 'glue' equations:
     ?drug_allergy_text.patient_id = ?admissions.patient_id

Continue?  (Y or N):  y

Translating logical formulas to SQL queries...

(:sql-tree
   (:select-items
      (:select-item (:attribute ADMISSIONS PATIENT_ID)
                    (:alias NIL)))
   (:relations
      (:relation ADMISSIONS))
```

```
   (:constraints
      (:constraint
         (:predicate <)
         (:literal-value 01-JAN-94)
         (:attribute ADMISSIONS ADMISSION_TIME))))

(:sql-tree
   (:select-items
      (:select-item (:attribute DRUG_ALLERGY_TEXT TEXT)
                    (:alias TEXT))
      (:select-item (:attribute DRUG_ALLERGY_TEXT PATIENT_ID)
                    (:alias PATIENT_ID)))
   (:relations
      (:relation DRUG_ALLERGY_TEXT))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute DRUG_ALLERGY_TEXT DRUG_ID)
         (:literal-value DRUGGO))))

Continue?  (Y or N):  y

Converting from minimized representation to actual representation ...

(:sql-tree
   (:select-items
      (:select-item (:attribute ADMISSIONS PATIENT_ID)
                    (:alias NIL)))
   (:relations
      (:relation ADMISSIONS))
   (:constraints
      (:constraint
         (:predicate <)
         (:literal-value 01-JAN-94)
         (:attribute ADMISSIONS ADMISSION_TIME))))

(:sql-tree
   (:select-items
      (:select-item (:attribute DRUG_ALLERGY TEXT)
                    (:alias TEXT))
      (:select-item (:attribute DRUG_ALLERGY PATIENT_ID)
                    (:alias PATIENT_ID)))
   (:relations
      (:relation DRUG_ALLERGY))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute DRUG_ALLERGY DRUG_ID)
```

```
        (:literal-value DRUGGO))))

Continue?  (Y or N):  y


Writing SQL queries on remote databases to files "demo-3-out-1.sql"
  and "demo-3-out-2.sql"...
Query on HOSPITAL_DATABASE_1 has been written to file "demo-3-out-1.sql"
Query on HOSPITAL_DATABASE_2 has been written to file "demo-3-out-2.sql"

Continue?  (Y or N):  y

Joining tables in files "demo-3-in-1.tbl" and "demo-3-in-2.tbl" for use as
  response to original query
Will write result to "demo-3-out.tbl" ...

Table read from file "demo-3-in-1.tbl", and file deleted
Table read from file "demo-3-in-2.tbl", and file deleted
Joined table has been written to file "demo-3-out.tbl"
```

Now, the fourth and final phase of the demonstration.

```
Reading SQL from file "demo-4-in.sql" ...
```

This time, the query is from a slightly modified Clinic database that uses specially-generated patient IDs rather than SSNs. The correlation between IDs and SSNs is stored in another restricted-access Clinic database, which the issuer of this query has permission to access.

```
Abstract syntax tree for the SQL form is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY DRUG_NAME)
                    (:alias NIL))
      (:select-item (:attribute NOTES TEXT)
                    (:alias NIL)))
   (:relations
      (:relation NOTES)
      (:relation PATIENT_ALLERGY))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute NOTES NOTE_ID)
         (:attribute PATIENT_ALLERGY NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
         (:literal-value P11111))))
```

```
Continue?  (Y or N):  y
Filling in omitted attributes and aliases in SQL tree ...
Completed abstract syntax tree for the SQL form is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY DRUG_NAME)
                    (:alias DRUG_NAME))
      (:select-item (:attribute NOTES TEXT)
                    (:alias TEXT)))
   (:relations
      (:relation NOTES)
      (:relation PATIENT_ALLERGY))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute NOTES NOTE_ID)
         (:attribute PATIENT_ALLERGY NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
         (:literal-value P11111))))
Continue?  (Y or N):  y
Converting query to minimized representation ...
Minimized abstract syntax tree is:
(:sql-tree
   (:select-items
      (:select-item (:attribute PATIENT_ALLERGY DRUG_NAME)
                    (:alias DRUG_NAME))
      (:select-item (:attribute NOTES_TEXT TEXT)
                    (:alias TEXT)))
   (:relations
      (:relation NOTES)
      (:relation PATIENT_ALLERGY)
      (:relation NOTES_TEXT))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute NOTES NOTE_ID)
         (:attribute PATIENT_ALLERGY NOTE_ID))
      (:constraint
         (:predicate =)
         (:attribute PATIENT_ALLERGY PATIENT_ID)
         (:literal-value P11111))
      (:constraint
         (:predicate =)
         (:attribute NOTES NOTE_ID)
         (:attribute NOTES_TEXT NOTE_ID))))
Continue?  (Y or N):  y
```

```
Translating SQL query to logical formula ...
Logical form of query is:
(E ?patient_allergy.note_id)
(E ?patient_allergy.patient_id)
(E ?notes.note_id)
(E ?notes_text.note_id)
   \/{ /\{  Notes(?notes.note_id),
            Patient_Allergy(
              ?patient_allergy.patient_id,
              ?patient_allergy.drug_name,
              ?patient_allergy.note_id),
            Notes_Text(?notes_text.note_id, ?notes_text.text),
            ?notes.note_id = ?patient_allergy.note_id,
            ?patient_allergy.patient_id = p11111,
            ?notes.note_id = ?notes_text.note_id}}
where the free variables of the formula are associated with query attribute
aliases as follows:
      ?PATIENT_ALLERGY.DRUG_NAME  <--> DRUG_NAME
      ?NOTES_TEXT.TEXT  <--> TEXT
Continue?  (Y or N):  y
Simplifying logical formula ...
Simplified formula is:
(E ?notes.note_id)
(E ?patient_allergy.note_id)
(E ?patient_allergy.patient_id)
   \/{ /\{  Patient_Allergy(
              ?patient_allergy.patient_id,
              ?patient_allergy.drug_name,
              ?patient_allergy.note_id),
            Notes_Text(?notes.note_id, ?notes_text.text),
            ?notes.note_id = ?patient_allergy.note_id,
            ?patient_allergy.patient_id = p11111}}
Continue?  (Y or N):  y
Attempting to derive queries on remote databases ...
Attempting to derive query on HOSPITAL_DATABASE ...
Writing query to file "demo-4-out-1.sql" ...
```

In order to obtain a query on Hospital database, the ID 'P11111' must be transformed into a SSN. So the mediator issues an auxiliary query to obtain this patient's SSN. The rule being used that causes the query to be issued is
$?patient\_allergy.patient\_id = k \longrightarrow ?patient\_allergy.patient\_id = k'$ *(k a constant), where k' is the result returned when the query*

```
    SELECT SSN
   FROM   PERSONAL_DATA
   WHERE  PATIENT_ID = k;
```

is run on the restricted-access Clinic database.

```
Query written to file "demo-4-out-1.sql"
```

```
Table read from file "demo-4-in-1.tbl", and file deleted
```

The mediator now knows that P11111's SSN is 123-45-6789.

```
Succeeded!
Logical form of derived query is:
(E ?drug_allergy_text.patient_id)
   \/{ /\{  ?drug_allergy_text.patient_id = 123-45-6789,
            Drug_Allergy_Text(
              ?drug_allergy_text.patient_id,
              ?drug_allergy_text.drug_id,
              ?drug_allergy_text.text)}}
where the variables in the two formulas are associated as follows:
      ?PATIENT_ALLERGY.PATIENT_ID  <--> ?DRUG_ALLERGY_TEXT.PATIENT_ID
      ?PATIENT_ALLERGY.DRUG_NAME  <--> ?DRUG_ALLERGY_TEXT.DRUG_ID
      ?NOTES_TEXT.TEXT  <--> ?DRUG_ALLERGY_TEXT.TEXT
Continue?  (Y or N):  y
Translating logical formula to SQL query ...
(:sql-tree
   (:select-items
      (:select-item (:attribute DRUG_ALLERGY_TEXT TEXT)
                    (:alias TEXT))
      (:select-item (:attribute DRUG_ALLERGY_TEXT DRUG_ID)
                    (:alias DRUG_NAME)))
   (:relations
      (:relation DRUG_ALLERGY_TEXT))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute DRUG_ALLERGY_TEXT PATIENT_ID)
         (:literal-value 123-45-6789))))
Continue?  (Y or N):  y
Converting from minimized representation to actual representation ...
(:sql-tree
   (:select-items
      (:select-item (:attribute DRUG_ALLERGY TEXT)
                    (:alias TEXT))
      (:select-item (:attribute DRUG_ALLERGY DRUG_ID)
                    (:alias DRUG_NAME)))
   (:relations
      (:relation DRUG_ALLERGY))
   (:constraints
      (:constraint
         (:predicate =)
         (:attribute DRUG_ALLERGY PATIENT_ID)
         (:literal-value 123-45-6789))))
Continue?  (Y or N):  y
Writing SQL query on remote database to file "demo-4-out-2.sql"...
```

```
Query has been written to file "demo-4-out-2.sql"

Continue?  (Y or N):  y
Converting table in file "demo-4-in-2.tbl" for use as response to
  original query
Will write result to "demo-4-out.tbl" ...
Table read from file "demo-4-in-2.tbl", and file deleted
Converted table has been written to file "demo-4-out.tbl"
```

This completes the last phase of the demonstration, so we exit the mediator.

C-c

```
>>Break: Keyboard interrupt

LUCID:%SLEEP:
:C  0: Return from Break

-> (quit)
%
```

Bibliography

[1] Y. Arens and C. Knoblock. Planning and reformulating queries for semantically-modeled multidatabase systems. In *Proceedings of the First International Conference on Information and Knowledge Management*, 1992.

[2] T. Barsalou and D. Gangopadhyay. M(DM): An open framework for interoperation of multimodel multidatabase systems. In *Proceedings of the Eighth International Conference on Data Engineering*, 1992.

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[4] B. W. Beach. Connecting software components with declarative glue. In *Proceedings of the Fourteenth International Conference on Software Engineering*, pages 120–137, 1992.

[5] R. J. Brachman. The future of knowledge representation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1082–1092, 1990.

[6] J.A. Bull, L. Gong, and K.R. Sollins. Towards Security in an Open Systems Federation. In *Proceedings of the European Symposium on Research in Computer Security*, volume 648 of *Lecture Notes in Computer Science*, pages 3–20, Toulouse, France, November 1992. Springer-Verlag.

[7] R. K. Burns. Referential secrecy. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 133–142, 1990.

[8] R. K. Burns. Integrity and secrecy: Fundamental conflicts in the database environment. In *Proceedings of the Third RADC Database Security Workshop, Technical Report MTP 385, MITRE*, pages 37–40, 1991.

[9] J. Chomicki and W. Litwin. Declarative definition of object-oriented multidatabase mappings. In M. Ozsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*. Morgan Kaufmann, 1993.

[10] L. DeMichiel. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. *IEEE Transactions on Knowledge and Data Engineering*, 1(4):485–493, December 1989.

[11] W. Du, R. Krishnamurthy, and M. Shan. Query optimization in a heterogeneous DBMS. In *Proceedings of the Eighteenth International Conference on Very Large Data Bases*, pages 227–291, 1992.

[12] R. Fagin. Horn clauses and database dependencies. *Journal of the ACM*, 29(4):952–985, October 1982.

[13] H. Gallaire, J. Minker, and J.-M. Nicolas. Logic and databases: A deductive approach. *ACM Computing Surveys*, 16(2):153–185, June 1984.

[14] D. Gangopadhyay and T. Barsalou. On the semantic equivalence of heterogeneous representations in multimodel multidatabase systems. *ACM SIGMOD Record*, 20(4), December 1991.

[15] M. L. Ginsberg. Knowledge interchange format: The KIF of death. *AI Magazine*, 12(3):57–63, 1991.

[16] J. Glasgow, G. MacEwen, and P. Panangaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10(3):226–264, August 1992.

[17] L. Gong and X. Qian. The complexity and composability of secure interoperation. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 190–200, 1994.

[18] J. T. Haigh, R. C. O'Brien, and D. J. Thomsen. The LDV secure relational DBMS model. In S. Jajodia and C. E. Landwehr, editors, *Database Security, IV: Status and Prospects*, pages 265–279. North-Holland, 1991.

[19] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[20] H. H. Hosmer. Integrating security policies. In *Proceedings of the Third RADC Database Security Workshop, Technical Report MTP 385, MITRE*, pages 169–173, 1991.

[21] S. Jajodia and R. Sandhu. Polyinstantiation integrity in multilevel relations. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 104–115, 1990.

[22] S. Jajodia and R. Sandhu. A novel decomposition of multilevel relations into single-level relations. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 300–313, 1991.

[23] W. Kent. Solving domain mismatch and schema mismatch problems with an object-oriented database programming language. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, 1991.

[24] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 393–402, 1992.

[25] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 40–49, 1991.

[26] B.W. Lampson. Protection. In *Proceedings of the 5th Princeton Symposium on Information Sciences and Systems*, Princeton University, March 1971. Reprinted in ACM Operating Systems Review, 8(1):18–24, January, 1974.

[27] C. E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[28] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987.

[29] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. The Seaview security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, June 1990.

*[30] T. F. Lunt, P. G. Neumann, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. Secure distributed data views: Security policy and interpretation for DBMS for a class A1 DBMS. Technical Report RADC-TR-89-313, Vol. 1, Rome Air Development Center, Air Force Systems Command, December 1989.

[31] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming*, volume 2. Addison-Wesley, 1990.

*Although this limited document is referenced, no limited information has been extracted. Distribution authorized to DOD & DOD contractors only; software documentation; Dec 89.

[32] C. Meadows and S. Jajodia. Integrity versus security in multilevel secure databases. In C. E. Landwehr, editor, *Database Security: Status and Prospects*, pages 89–101. North-Holland, 1988.

[33] NCSC. *Trusted Network Interpretation Environments Guideline.* (U.S.) National Computer Security Center, August 1990. NCSC-TG-011 version-1.

[34] R. Neches, R. E. Fikes, T. Finin, T. Gruber, R. S. Patil, T. Senator, and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, 1991.

[35] J.-M. Nicolas and H. Gallaire. Data base: Theory vs. interpretation. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 33–54. Plenum Press, 1978.

[36] G. Pernul. Canonical security modeling for federated databases. In *Proceedings of the IFIP WG 2.6 Conference on Semantics of Interoperable Database Systems*, 1992.

[37] X. Qian. A model-theoretic semantics of the multilevel secure relational model. Technical Report SRI-CSL-93-06, Computer Science Laboratory, SRI International, November 1993.

[38] X. Qian. Semantic interoperation via intelligent mediation. In *Proceedings of the Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, pages 228–231, 1993.

[39] X. Qian. Inference channel-free integrity constraints in multilevel relational databases. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 158–167, 1994.

[40] X. Qian. Query folding. Technical Report SRI-CSL-95-09, Computer Science Laboratory, SRI International, June 1995.

[41] X. Qian and T. F. Lunt. Tuple-level vs. element-level classification. In B. M. Thuraisingham and C. E. Landwehr, editors, *Database Security, VI: Status and Prospects*, pages 301–315. North-Holland, 1993.

[42] L. Raschid, Y. Chang, and B.J. Dorr. Query mapping and transformation techniques for problem solving with multiple knowledge servers. In *Proceedings of the Second International Conference on Information and Knowledge Management*, 1993.

[43] R. Sandhu and S. Jajodia. Eliminating polyinstantiation securely. *Computers & Security*, 11:547–562, 1992.

[44] R. Sandhu, S. Jajodia, and T. F. Lunt. A new polyinstantiation integrity constraint for multilevel relations. In *Proceedings of the Third IEEE Workshop on Computer Security Foundations*, pages 159–165, 1990.

[45] R. S. Sandhu. The typed access matrix model. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, pages 122–136, 1992.

[46] M. Satyanarayanan. Integrating Security in a Large Distributed System. *ACM Transactions on Computer System*, 7(3):247–280, August 1989.

[47] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems*, 19(2):254–290, June 1994.

[48] A. Sheth. Semantic issues in multidatabase systems. *ACM SIGMOD Record*, 20(4), December 1991.

[49] A. Sheth and S. Gala. Attribute relationships: An impediment to automating schema integration. In *Proceedings of the Workshop on Heterogeneous Database Systems*, 1989.

[50] A. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In *Proceedings of the IFIP TC2/WG2.6 Conference on Semantics of Interoperable Database Systems*. Elsevier Scientific Publishers, 1992.

[51] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[52] G. Smith. Modeling security-relevant data semantics. *IEEE Transactions on Software Engineering*, 17(11):1195–1203, November 1991.

[53] K. Smith and M. Winslett. Entity modeling in the MLS relational model. In *Proceedings of the Eighteenth International Conference on Very Large Data Bases*, pages 199–210, 1992.

[54] T. Su and G. Ozsoyoglu. Controlling FD and MVD inferences in multilevel relational database systems. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):474–485, December 1991.

[55] B. M. Thuraisingham. A nonmonotonic typed multilevel logic for multilevel secure database/knowledge-base management systems. In *Proceedings of the Fourth IEEE Workshop on Computer Security Foundations*, pages 127–138, 1991.

[56] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1. Computer Science Press, 1988.

[57] J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 2. Computer Science Press, 1989.

[58] *Trusted Network Interpretation*. U.S. National Computer Security Center, July 1987. NCSC-TG-005 version-1.

[59] G. Wiederhold. Views, objects, and databases. *IEEE Computer*, 19(12):37–44, December 1986.

[60] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[61] G. Wiederhold. Model-free optimization. In *Proceedings of the DARPA Software Technology Conference*, pages 83–96, 1992.

[62] S. R. Wiseman. Control of confidentiality in databases. *Computers & Security*, 9(6):529–537, October 1990.

*U.S. GOVERNMENT PRINTING OFFICE: 1996-710-126-20116

DISTRIBUTION LIST

addresses	number of copies
JOSEPH GIORDANO RL/C3AB 525 BROOKS RD. ROME NY 13441-4505	5
XIALOEI QIAN SRI INTERNATIONAL 333 RAVENSWOOD AVENUE MENLO PARK CA 94025	5
ROME LABORATORY/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
AGCS, INC. ATTN: RAE BURNS 91 MONTVALE AVENUE STONEHAM, MA 02180-3616	1
NAVAL RESEARCH LABORATORY ATTN: OLIVER COSTICH CODE 5542 WASHINGTON D.C. 20375-5337	1
NAVAL RESEARCH LABORATORY ATTN: JUDITH FROSCHER CODE 5542 WASHINGTON D.C. 20375-5337	1

SECURE COMPUTING CORPORATION 1
ATTN: THOMAS HAIGH
2675 LONG LAKE ROAD
ROSEVILLE, MN 55113

MITRE CORPORATION 1
ATTN: WILLIAM R. HERNDON
7525 COLSHIRE DRIVE
MAIL STOP Z231
MCLEAN, VA 22102

SAN JOSE STATE UNIVERSITY 1
MATHEMATICS & COMPUTER SCIENCE
ATTN: T.Y. LIN
SAN JOSE, CA 95192

ARPA/CSTO 1
ATTN: TERESA LUNT
3701 N. FAIRFAX DRIVE
ARLINGTON, VA 22203

NATIONAL SECURITY AGENCY 1
ATTN: DONALD MARKS
R23
9800 SAVAGE ROAD
FORT MEADE, MD 20755-6000

MITRE CORPORATION 1
ATTN: CATHERINE D. MCCOLLUM
7525 COLSHIRE DRIVE
MAIL STOP Z231
MCLEAN, VA 22102

ORACLE CORPORATION 1
ATTN: LOUANNA NOTARGIACOMO
196 VAN BUREN STREET
HERNDON, VA 22070

SECURE COMPUTING CORPORATION 1
ATTN: DICK O'BRIEN
2675 LONG LAKE ROAD
ROSEVILLE, MN 55113

INFOSYSTEMS TECHNOLOGY, INC. 1
ATTN: JAMES P. O'CONNOR
1835 ALEXANDER BELL DRIVE
SUITE 230
RESTON, VA 22091

SRI INTERNATIONAL
ATTN: XIAOLEI QIAN
333 RAVENSWORTH AVENUE
MENLO PARK, CA 94025

1

GEORGE MASON UNIVERSITY
DEPT OF INFO & SOFTWARE SYS ENG
ATTN: RAVI SANDHU
FAIRFAX, VA 22030-4444

1

ARCA SYSTEMS/ATTN: MARVIN SCHAEFER
1 COMMERCE CENTER
10320 LITTLE PATUXENT PARKWAY
SUITE 1005
COLUMBIA, MD 21044-3312

1

MITRE CORPORATION
ATTN: KENNETH P. SMITH
7525 COLSHIRE DRIVE
MAIL STOP Z231
MCLEAN, VA 22102

1

SECURE COMPUTING CORPORATION
ATTN: DAN THOMSON
2675 LONG LAKE ROAD
ROSEVILLE, MN 55113

1

ONTOS
ATTN: SANDRA WADE
THREE BURLINGTON WOODS
BURLINGTON, MA 01803

1

DEFENSE RESEARCH AGENCY
ATTN: SIMON WISEMAN
ST. ANDREWS ROAD
MALVERN, WORCESTERSHIRE
WR14 EPS, UK

1

ARCA SYSTEMS
ATTN: JACK WOOL
ESC/ENS
BLDG 1704, RM 104
HANSCOM AFB MA 01731-5000

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.